

175 PTAS
95
mi compuTer
CURSO PRACTICO DEL ORDENADOR PERSONAL,
EL MICRO Y EL MINIORDENADOR
Editorial Delta, S.A.

MI COMPUTER CURSO PRACTICO
DEL ORDENADOR PERSONAL, EL MICRO Y EL MINIORDENADOR

Publicado por Editorial Delta, S.A., Barcelona

Volumen VIII-Fascículo 95

Director: José Mas Godayol
Director editorial: Gerardo Romero
Jefe de redacción: Pablo Parra
Coordinación editorial: Jaime Mardones
Francisco Martín
Asesor técnico: Ramón Cervelló

Redactores y colaboradores: G. Jefferson, R. Ford, F. Martín, S. Tarditti, A. Cuevas, F. Blasco
Para la edición inglesa: R. Pawson (editor), D. Tebbutt (consultant editor), C. Cooper (executive editor), D. Whelan (art editor), Bunch Partworks Ltd. (proyecto y realización)

Realización gráfica: Luis F. Balaguer

Redacción y administración:
Aribau, 185, 1.°, 08021 Barcelona
Tel. (93) 209 80 22 -

MI COMPUTER, *Curso práctico del ordenador personal, el micro y el miniordenador*, se publica en forma de 96 fascículos de aparición semanal, encuadernables en ocho volúmenes. Cada fascículo consta de 20 páginas interiores y sus correspondientes cubiertas. Con el fascículo que completa cada uno de los volúmenes, se ponen a la venta las tapas para su encuadernación.
El editor se reserva el derecho de modificar el precio de venta del fascículo en el transcurso de la obra, si las circunstancias del mercado así lo exigieran.

ISBN: 84-85822-83-8 (fascículo) 84-7598-067-2 (tomo 7)
84-85822-82-X (obra completa)
Depósito Legal: B. 52-84

Fotocomposición: Tecfa, S.A., Pedro IV, 160, Barcelona-5
Impresión: Cayfosa, Santa Perpètua de Mogoda (Barcelona) 208511
Impreso en España-Printed in Spain-Noviembre 1985

Editorial Delta, S.A., garantiza la publicación de todos los fascículos que componen esta obra.

Distribuye para España: Marco Ibérica, Distribución de Ediciones, S.A., Carretera de Irún, km 13,350. Variante de Fuencarral, 28034 Madrid.
Distribuye para Colombia: Distribuidoras Unidas, Ltda., Transversal 93, n.° 52-03, Bogotá D.E.
Distribuye para México: Distribuidora Intermex, S.A., Lucio blanco, n.° 435, Col. San Juan Tlihuaca, Azcapotzalco, 02400, México D.F.
Distribuye para Venezuela: Distribuidora Continental, S.A., Edificio Bloque Dearmas, final Avda. San Martín con final Avda. La Paz, Caracas 1010.

Pida a su proveedor habitual que le reserve un ejemplar de MI COMPUTER. Comprando su fascículo todas las semanas y en el mismo quiosco o librería, Vd. conseguirá un servicio más rápido, pues nos permite realizar la distribución a los puntos de venta con la mayor precisión.

Servicio de suscripciones y atrasados (sólo para España)

Las condiciones de suscripción a la obra completa (96 fascículos más las tapas, guardas y transferibles para la confección de los 8 volúmenes) son las siguientes:

a) Un pago único anticipado de 19 425 ptas. o bien 8 pagos trimestrales anticipados y consecutivos de 2 429 ptas. (sin gastos de envío).

b) Los pagos pueden hacerse efectivos mediante ingreso en la cuenta 6.850.277 de la Caja Postal de Ahorros y remitiendo a continuación el resguardo o su fotocopia a Editorial Delta, S. A. (Aribau, 185, 1.°, 08021 Barcelona), o también con talón bancario remitido a la misma dirección.

c) Se realizará un envío cada 12 semanas, compuesto de 12 fascículos y las tapas para encuadernarlos.

Los fascículos atrasados pueden adquirirse en el quiosco o librería habitual. También pueden recibirse por correo, con incremento del coste de envío, haciendo llegar su importe a Editorial Delta, S.A., en la forma establecida en el apartado b).

No se efectúan envíos contra reembolso.





Tony Sleep

Reconocer un gesto

En esta ocasión analizaremos un método que explora patrones en busca de rasgos significativos

¿Cuál es la expresión de tu rostro?
El sistema WISARD, de Igor Aleksander y desarrollado en el Imperial College de Londres, utiliza una cuadrícula de 512 por 512 y un megabyte de RAM para poder reconocer patrones visualizados en una pantalla de televisión. Es tan sensible que se le puede enseñar a distinguir entre rostros sonrientes y rostros serios

Los llamados sistemas de reconocimiento de patrones *de abajo arriba* intentan extraer rasgos significativos de una imagen dada y reducirla a un patrón más simple y de un mayor nivel de abstracción. Un método para hallar tales rasgos en un patrón consiste en utilizar operadores locales para explorarlo. Cada operador local explora una pequeña superficie cada vez y multiplica las intensidades de gris de los puntos bajo el operador por factores de estimación. Estos factores están diseñados para producir marcadores elevados cuando se localiza el rasgo buscado. Mediante el empleo de numerosos operadores locales diferentes, se pueden extraer las posiciones de rasgos importantes, tales como líneas horizontales, verticales y oblicuas o parches de luz y oscuridad.

El programa WISARD, de Igor Aleksander, que describimos en el capítulo anterior, utiliza el principio de *generación de dirección de memoria* desde un patrón con el objeto de recordar el patrón y reconocerlo posteriormente.

La gran ventaja del WISARD reside en que todos los bancos de RAM se direccionan en paralelo. Por tanto, puede trabajar a gran velocidad con imágenes de televisión muy nítidas. El programa en BASIC que ofrecemos como ejemplo simula secuencialmente un sistema de este tipo. Es mucho mas lento, pero demuestra claramente los principios involucrados.

Para su labor de diferenciación el sistema utiliza séxtuplos (son menos potentes que los óctuplos pero ahorran espacio) y requiere 20 480 bits de RAM, o 2 560 bytes. Ello se debe a que hay 40 séxtuplos, cada uno de los cuales puede estar en 64 estados, y, por tanto, direccionar un banco de RAM de 64 posiciones.

Cada posición contiene ocho bits, lo que permite al sistema diferenciar ocho distintas clases de entrada.

El programa posee dos fases: la primera es una etapa de entrenamiento en la cual se proporcionan al ordenador ejemplos de cada diferente clase de entrada. La segunda fase implica el reconocimiento de un patrón a través de la comparación con ejemplos aprendidos previamente. Por ejemplo, supongamos que durante la fase de entrenamiento el séxtuplo número 20 da como respuesta 110010 (50) cuando está presente el patrón de clase 4. En este caso, el cuarto bit de la posición 50 del banco de RAM 20 se establecerá en 1. Si durante la fase de reconocimiento el diferenciador vuelve a responder con otro 50, encontrará al cuarto bit establecido en

la dirección especificada, lo que tiende a confirmar la presencia del patrón número 4.

Para retener el patrón de datos se utiliza una serie de bytes que comienza en D, y toda la matriz se borra al principio de cada ejecución. Esto se consigue mediante la subrutina 1000. La subrutina 2000 permite al usuario «pintar» en la pantalla imágenes sencillas de 16 por 16, utilizando las teclas U, D, L y R para mover, y usando el cáracter * y la barra espaciadora para crear el patrón.

El programa se puede modificar de modo que el usuario pueda cargar y guardar la matriz D en disco o cinta. Tal como está, la primera vez que ejecute el programa, habrá de enseñarle una tarea de clasificación a partir de cero, de modo que, en este sentido, está en libertad de introducir cualquier mejora que considere necesaria. Por supuesto, lo ideal sería que el sistema estuviera unido al mundo exterior a través de un convertidor de analógico a digital fijado en algún tipo de cámara.

Observe que la asignación de pixels a los séxtuplos se realiza al azar, pero que se ha de poder repetir. La imagen del patrón inicialmente se retiene en una matriz bidimensional, I(,), utilizando un uno para representar un asterisco y un cero para representar un espacio. Durante la fase de entrenamiento se generan direcciones desde elementos de I(,) seleccionados al azar. El bit apropiado (según la clase que se esté aprendiendo) se establece en la línea 210. Durante la fase de reconocimiento se debe generar la misma secuencia de direcciones seleccionadas al azar, para poder comparar el patrón que se está comprobando con las clasificaciones aprendidas previamente.

Programa de reconocimiento de patrones

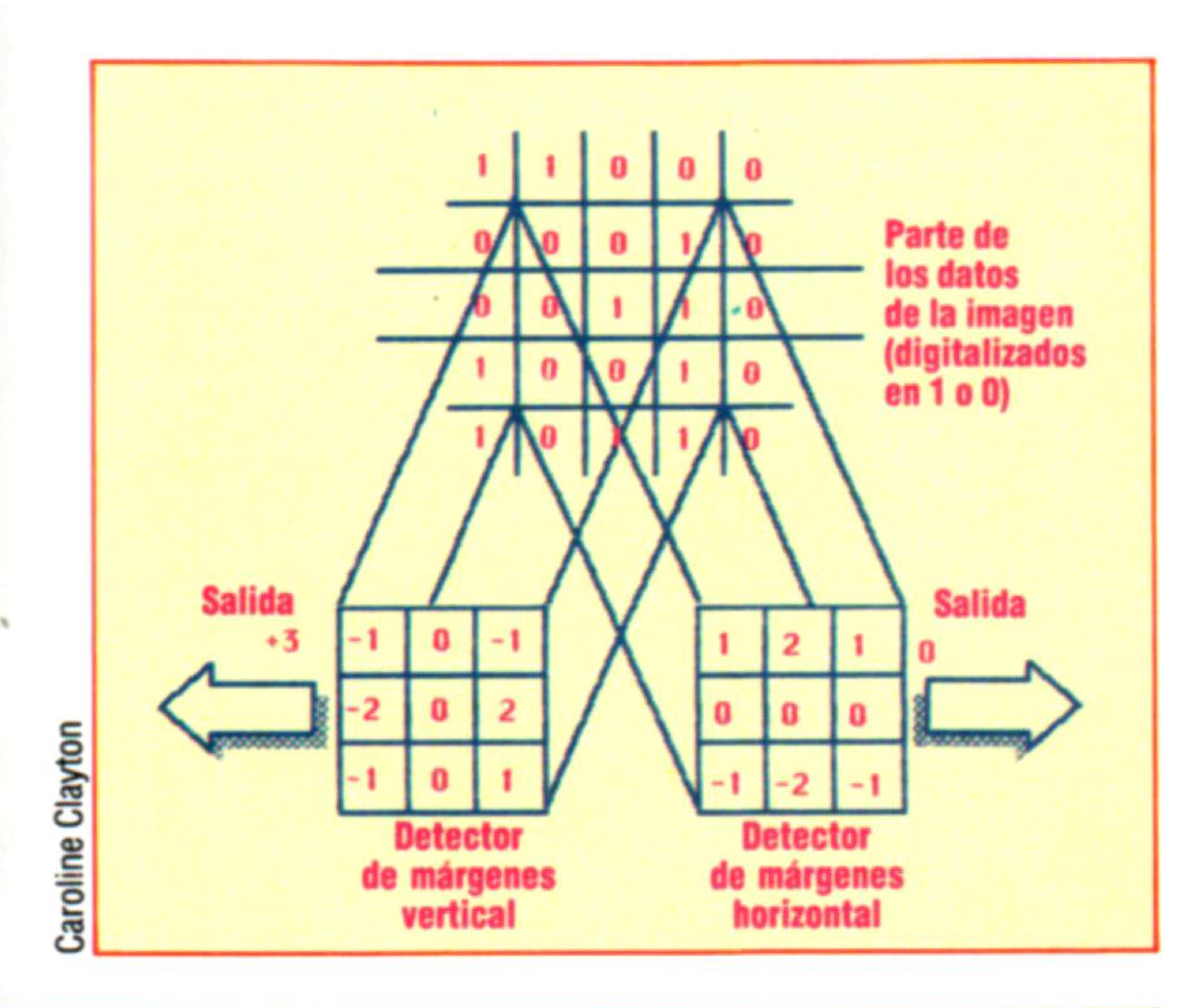
BBC Micro:

```
10 REM ****************************************************
30 REM ** RECONOCEDOR DE PATRONES **
40 REM ** TIPO WISARD PARA EL BBC **
50 REM ****************************************************
55 MODE 7
56 SX%=40: DS%=6: RB%=2^DS%
57 MEMO%=SX%*RB%
60 DIM I%(16,16), CS%(7)
64 DIM D% MEMO%
65 @%=4: REM formato o/p
66  REM D% es una matriz de bytes (BBC)
70  REM – – I% es matriz de imagen, D% es matriz direcciones
80 REM – – Nota: I% contiene 0 o 1
85 PRINT "Tamaño total de RAM = ";MEMO%;" bytes."
88 GOSUB 1000: REM limpiar matriz D%
90 REM – – Primero la fase de entrenamiento:
92 MOOD$="Entrenamiento"
95 INPUT "De que clase es (0..7) ",C%
96 IF C% <0 OR C%>7 THEN GOTO 90
100 FOR I%=1 TO 16:FOR J%=1 TO 16: I%(I%,J%)=0
101 NEXT:NEXT
110 REM – – Ahora tomar la imagen pintada por el usuario:
120 GOSUB 2000
130 R=RND(–1):REM valor al azar
140 FOR I%=1 TO SX%
150 A%=RB%*(I%–1)
155 REM A% es la direccion de base del banco de RAM.
160 FOR J%=0 TO (DS%–1)
170 R1%=INT(RND(1)*16)*1)
180 R2%=INT(RND(1)*16)*1)
190 A%=A%*I%(R1%,R2%)*2^J%
200 NEXT J%
202 REM R1% y R2% son coordenadas «aleatorias».
210 PROCdset(A%,C%)
220 NEXT I%
230 INPUT "Otra sesion de entrenamiento (S/N)?",A$;IF A$="S" THEN GO TO
    90
233 IF A$="s" THEN GOTO 90
235 MOOD$="Reconocimiento"
236 REM – – Ahora la Fase de Reconocimiento:
240 FOR I%=1 TO 16: FOR J%=1 TO 16
244 I%(I%,J%)=0
245 NEXT :NEXT
250 REM – – Obtener una imagen para clasificacion:
260 GOSUB 2000
270 R=RND(–1)
280 FOR C%=0 TO 7:CS%(C%)=0: NEXT
290 FOR I%=1 TO SX%
300 A%=RB%*(I%–1)
310 FOR J%=0 TO DS%–1
320 R1%=INT(RND(1)*16)+1
330 R2%=INT(RND(1)*16)+1
340 A%=A%*I%(R1%,R2%)*2^J%
350 NEXT
355 FOR C%=0 TO 7
360 IF DNdget(A%,C%) > 0 THEN CS%(C%)=CS%(C%)+1
370 NEXT C%
380 NEXT I%
382 CX%=0
385 FOR C%=0 TO 7
388 PRINT "La clase";C%;" posee un marcador de
    ";CS%(C%)/SX%*100;SPC(8)
390 IF CS%(C%)>CS%(CX%) THEN CX%=C%
400 NEXT C%
404 PRINT "Lo mas probable es que la clase sea la numero ";CX%
410 INPUT "Quieres clasificar otra imagen (S=Si) ";A$
420 IF A$="S" OR A$="s" THEN GOTO 240
440 PRINT: PRINT "Adios!"
444 END
999 :
1000 REM – – Inicializacion de memoria:
1010 FOR I%=0 TO MEMO%
1020 REM – – usa subindices de byte (?):
1030 ?(D%+I%)=0
1040 NEXT
1050 RETURN
1199 :
1200 DEF PROCdset(A%,C%)
1210 REM – – – – – – – – –
1220 ?(D%+A%) = (D%?A%) OR 2^C%
1230 ENDPROC
1240 REM establece un bit de la matriz D%
1244 :
1250 DEF FNdget(A%,C%)
1260 REM – – – – – –
1270 =(D%?A%) AND 2^C%
1280 REM extrae bit de la matriz D%
1288 REM A% es direccion; C% es 0..7
1290 :
2000 REM – – Rutina formacion-imagen:
2010 CLS:PRINT TAB(0,20);"(Usar: U, D, L, R, X * o espacio"
2012 PRINT "para definir la imagen.)"
2013 PRINT MOOD$; "fase."
2015 PRINT TAB(1,1);
2020 T%=0;A$="    "
2030 H%=1: V%=1
2040 REPEAT    C$=INKEY$(222)
2050 IF C$="U" THEN V%=V%–1
2060 IF C$="D" THEN V%=V%+1
2070 IF C$="L" THEN H%=H%–1
2080 IF C$="R" THEN H%=H%+1
2090 IF H% > 16 THEN H%=1
2095 IF H% < 1 THEN H%=16
2100 IF V% > 16 THEN V%=1
2105 IF V% < THEN V%=16
2110 IF C$="    " THEN T%=0:A$=C$
2120 IF C$="*" THEN T%=1:A$=C$
2130 I%(H%,V%)=T%
2140 PRINT TAB(H%,V%);A$;
2150 UNTIL C$="X"
2155 GOSUB 2200: REM Volver a visualizar
2160 RETURN
2170 :
2200 REM – – Rutina de visualizacion:
2210 FOR H%=1 TO 16
2220 FOR V%=1 TO 16
2230 IF I%(H%,V%)>0 THEN PRINT TAB(H%,V%);":" ELSE PRINT
     TAB(H%,V%);"    ";
2240 NEXT: NEXT
2250 PRINT TAB(0,17);
2260 RETURN
2270 :
2300 REM las instrucciones para "pintar" son:
2301 REM U & D para arriba y abajo;
2302 REM L & R para izquierda y derecha;
2303 REM * & espacio para On y Off;
2304 REM X para salir (imagen hecha).
```

Commodore 64:

```
10 REM **** RECONOCEDOR ****
20 REM **** DE PATRONES CBM 64 ****
30 FOR I=1 TO 25: DW$=DW$+CHR$(17):NEXT I
56 SX=40:DS=6:RB=2^DS
57 ME=SX*RB
60 DIM I(16,16),C(8)
64 D=12*4096:REM USAR $C000 PARA MATRIZ D
85 PRINT "TAMAÑO TOTAL DE RAM=";ME;" BYTES"
88 GOSUB 1000:REM LIMPIAR MATRIZ D
90 REM **** FASE DE ENTRENAMIENTO ****
92 M$="ENTRENAMIENTO"
95 INPUT"DE QUE CLASE ES (1...8)";C
96 IF C<1 OR C>8 THEN 95
100 FOR I=1 TO 16:FOR J=1 TO 16:I(I,J)=0
101 NEXT J:NEXT I
```



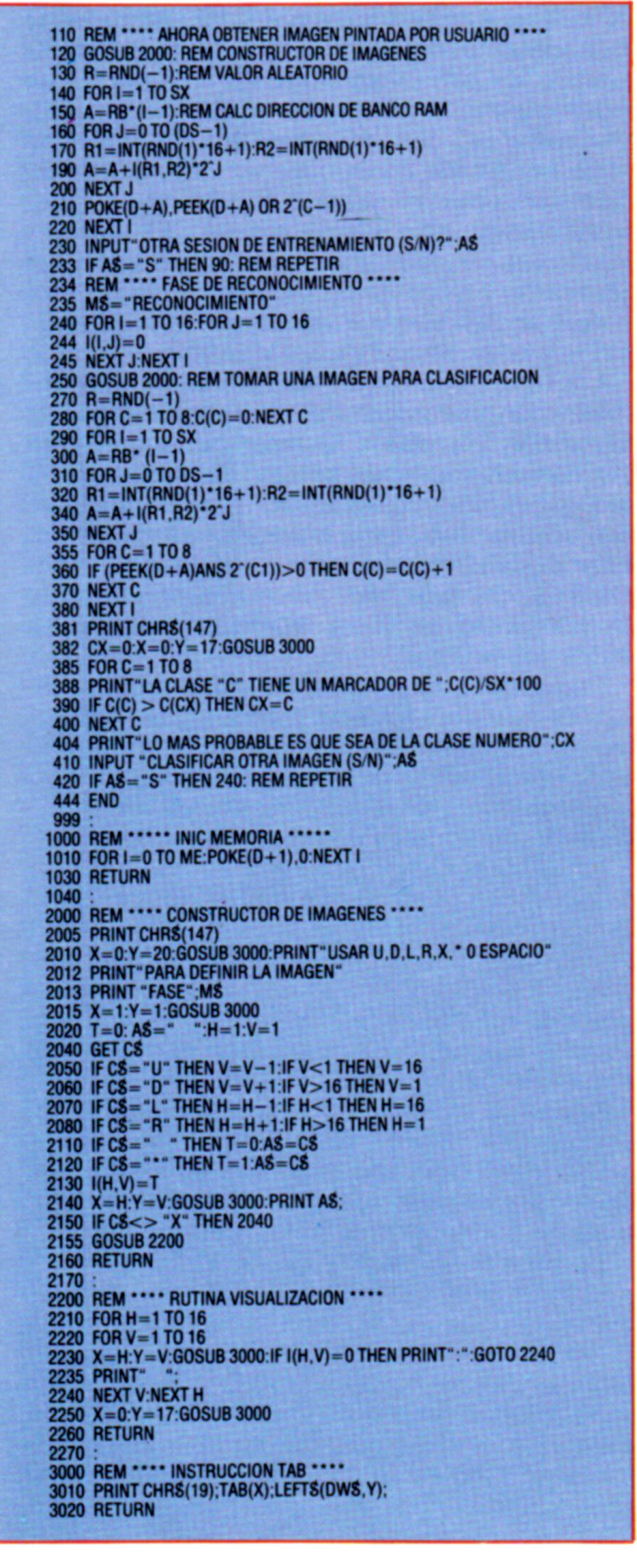


Caroline Clayton



Operadores locales
En la ilustración de la izquierda vemos un ejemplo del tipo de proceso de bajo nivel que puede emplear un sistema de reconocimiento de patrones de abajo arriba. Las cuadrículas de tres por tres se denominan *operadores locales* y exploran los datos de la imagen y registran marcadores elevados en las transiciones de bordes, dando, por lo tanto, un boceto primario de la imagen basado en líneas rectas. Los operadores se mueven a través de los datos de la imagen y cada número se multiplica por la intensidad de niveles de gris de la sección del patrón sobre la cual esté colocado. La suma de los productos da marcadores altos cuando el rasgo deseado está presente, y marcadores bajos en caso contrario. Cada uno de los nueve números de un operador local es un factor de estimación; el patrón de estas estimaciones determina los rasgos que se detectarán

Ojo óctuplo
Conformando la base del sistema WISARD para reconocimiento de patrones hay grupos de ocho pixels conectados arbitrariamente a ocho registros de bits. Cada óctuplo corresponde a un banco de 256 posiciones. Durante la fase de entrenamiento, el valor del óctuplo forma una dirección dentro de su banco de RAM, según el patrón de pixels, y en esta dirección se graba un 1. Si durante la fase de reconocimiento está presente el mismo patrón, se generará la misma dirección del banco de RAM y el 1 presente significará el reconocimiento



```
110 REM **** AHORA OBTENER IMAGEN PINTADA POR USUARIO ****
120 GOSUB 2000: REM CONSTRUCTOR DE IMAGENES
130 R=RND(-1):REM VALOR ALEATORIO
140 FOR I=1 TO SX
150 A=RB*(I-1):REM CALC DIRECCION DE BANCO RAM
160 FOR J=0 TO (DS-1)
170 R1=INT(RND(1)*16+1):R2=INT(RND(1)*16+1)
190 A=A+I(R1,R2)*2^J
200 NEXT J
210 POKE(D+A),PEEK(D+A) OR 2^(C-1))
220 NEXT I
230 INPUT"OTRA SESION DE ENTRENAMIENTO (S/N)?";A$
233 IF A$="S" THEN 90: REM REPETIR
234 REM **** FASE DE RECONOCIMIENTO ****
235 M$="RECONOCIMIENTO"
240 FOR I=1 TO 16:FOR J=1 TO 16
244 I(I,J)=0
245 NEXT J:NEXT I
250 GOSUB 2000: REM TOMAR UNA IMAGEN PARA CLASIFICACION
270 R=RND(-1)
280 FOR C=1 TO 8:C(C)=0:NEXT C
290 FOR I=1 TO SX
300 A=RB* (I-1)
310 FOR J=0 TO DS-1
320 R1=INT(RND(1)*16+1):R2=INT(RND(1)*16+1)
340 A=A+I(R1,R2)*2^J
350 NEXT J
355 FOR C=1 TO 8
360 IF (PEEK(D+A)ANS 2^(C1))>0 THEN C(C)=C(C)+1
370 NEXT C
380 NEXT I
381 PRINT CHR$(147)
382 CX=0:X=0:Y=17:GOSUB 3000
385 FOR C=1 TO 8
388 PRINT"LA CLASE "C" TIENE UN MARCADOR DE ";C(C)/SX*100
390 IF C(C) > C(CX) THEN CX=C
400 NEXT C
404 PRINT"LO MAS PROBABLE ES QUE SEA DE LA CLASE NUMERO";CX
410 INPUT "CLASIFICAR OTRA IMAGEN (S/N)";A$
420 IF A$="S" THEN 240: REM REPETIR
444 END
999 :
1000 REM ***** INIC MEMORIA *****
1010 FOR I=0 TO ME:POKE(D+1),0:NEXT I
1030 RETURN
1040 :
2000 REM **** CONSTRUCTOR DE IMAGENES ****
2005 PRINT CHR$(147)
2010 X=0:Y=20:GOSUB 3000:PRINT"USAR U,D,L,R,X,* 0 ESPACIO"
2012 PRINT"PARA DEFINIR LA IMAGEN"
2013 PRINT "FASE";M$
2015 X=1:Y=1:GOSUB 3000
2020 T=0: A$="    ":H=1:V=1
2040 GET C$
2050 IF C$="U" THEN V=V-1:IF V<1 THEN V=16
2060 IF C$="D" THEN V=V+1:IF V>16 THEN V=1
2070 IF C$="L" THEN H=H-1:IF H<1 THEN H=16
2080 IF C$="R" THEN H=H+1:IF H>16 THEN H=1
2110 IF C$="    " THEN T=0:A$=C$
2120 IF C$="*" THEN T=1:A$=C$
2130 I(H,V)=T
2140 X=H:Y=V:GOSUB 3000:PRINT A$;
2150 IF C$<> "X" THEN 2040
2155 GOSUB 2200
2160 RETURN
2170 :
2200 REM **** RUTINA VISUALIZACION ****
2210 FOR H=1 TO 16
2220 FOR V=1 TO 16
2230 X=H:Y=V:GOSUB 3000:IF I(H,V)=0 THEN PRINT":":GOTO 2240
2235 PRINT"    ";
2240 NEXT V:NEXT H
2250 X=0:Y=17:GOSUB 3000
2260 RETURN
2270 :
3000 REM **** INSTRUCCION TAB ****
3010 PRINT CHR$(19);TAB(X);LEFT$(DW$,Y);
3020 RETURN
```

Complementos al BASIC

Spectrum:

El Spectrum no soporta las instrucciones sobre bits AND y OR. En consecuencia, hemos de escribir cortas rutinas en código máquina para llevar a cabo estas operaciones. Éstas se ofrecen en forma de un cargador de BASIC. Introduzca estos cambios en la versión para el Commodore 64:

```
30 CLEAR 49999:GOSUB 4000

64 LET D=50018

130 RANDOMIZE 1

210 LET X1=PEEK(D+A):LET Y1=2 ↑ (C-1):
    GOSUB 6000

215 POKE(D+A),R1

270 RANDOMIZE 1

360 LET X1=PEEK(D+A):LET Y1=2 ↑ (C-1):
    GOSUB 5000

365 IF R1>0 THEN C(C)=C(C)+1

381 CLS

2005 CLS

3010 PRINT AT Y,X;

4000 FOR I=50000 TO 50017
4010 READ A:POKE I,A:NEXT I
4020 RETURN
4030 DATA 62,0,14,0,161,6,0,79,201
4040 DATA 62,0,14,0,177,6,0,79,201

5000 REM **** AND ****
5010 POKE 50001,X1:POKE 50003,Y1
5020 LET R1=USR 50000:RETURN

6000 REM **** OR ****
6010 POKE 50010,X1:POKE 50012,Y1
6020 LET R1=USR 50009:RETURN
```

Ventajosa comparación

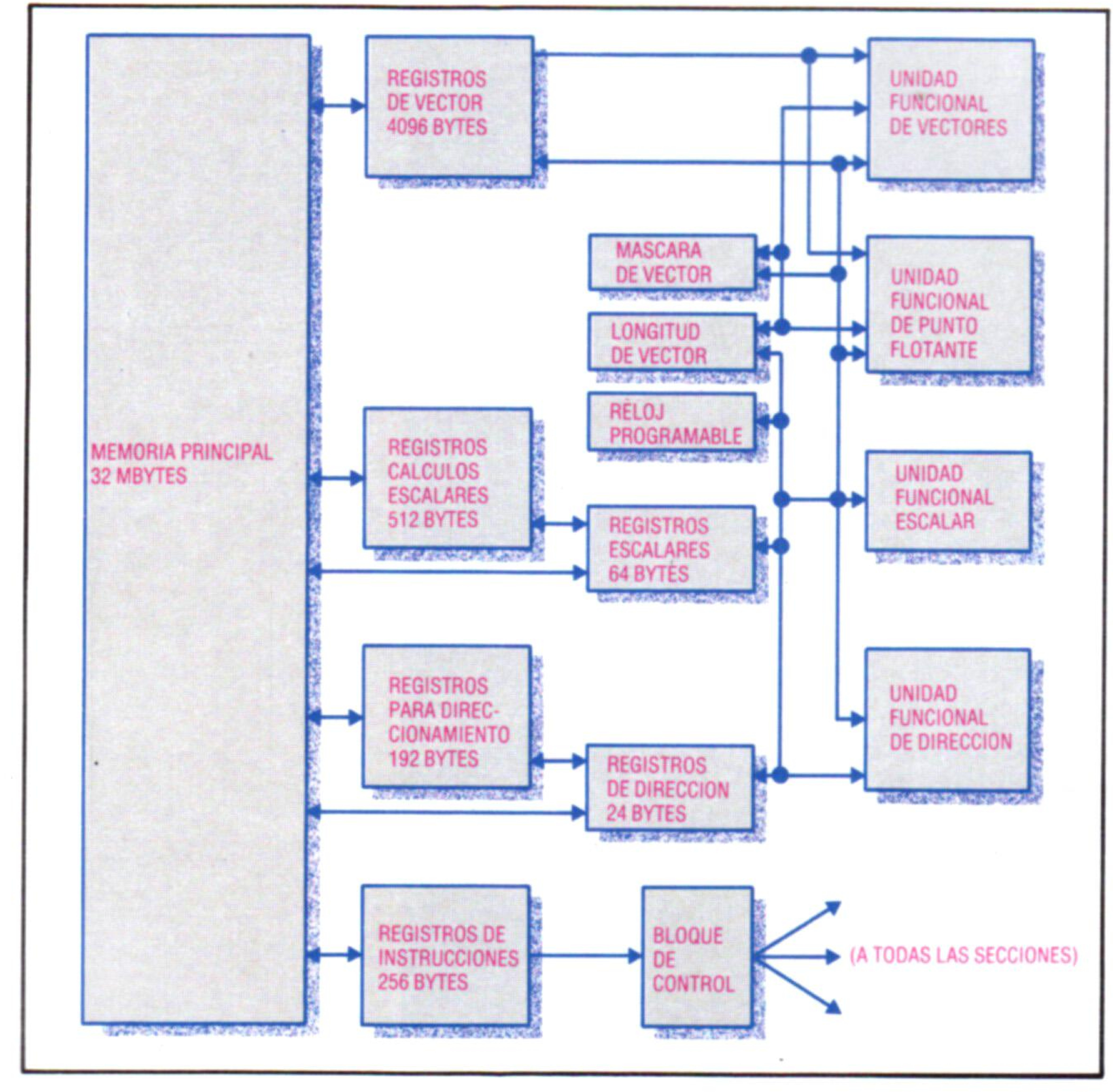
Centraremos nuestra atención en el Cray-1, superordenador capaz de ejecutar aplicaciones de proceso masivo de información

La forma del favorito
El Cray-1 (arriba) ofrece al usuario un enorme poder de proceso bajo el control de un ordenador frontal como un IBM, un DEC u otro similar; la arquitectura del Cray-1 (abajo) configura 32 Mbytes de memoria principal y 4 888 bytes de espacio para registros, contenidos en un sistema de 3 400 placas de circuito impreso separadas y más de 90 km de cable para conectarlas

La «potencia» de un ordenador, en términos llanos, es el producto de su tamaño de palabra, su velocidad de transferencia de datos, el tamaño de su memoria principal y la velocidad de ciclos de máquina de su unidad central de proceso. En los microordenadores, las funciones principales de la CPU tradicional están empaquetadas en un único microprocesador. Entre estos microprocesadores se incluyen el difundido Z80, el Intel 8088 y el 8086 y el Motorola M68000. Todos utilizan tecnología MOS (*metal oxide semiconductor*) para los circuitos lógicos y la memoria *on-chip*. Los datos se transfieren y se procesan en paralelo ya sea de ocho en ocho o bien de 16 en 16. En los microprocesadores las frecuencias de reloj van de 1 MHz a 12 MHz.

Las especificaciones como éstas, aun siendo notables, hacen que los ordenadores basados en microprocesador sean incapaces de afrontar la colosal cantidad de proceso de datos que requieren aplicaciones tales como el proceso de imágenes, la dinámica de fluidos y el pronóstico meteorológico.

Supongamos que usted está haciendo una película con gráficos animados generados por ordenador, y que la imagen requiere una resolución de 6 000 por 6 000 pixels y 24 fotogramas por segundo. Puesto que los gráficos son animados, la posición de cada punto de la imagen puede moverse entre los fotogramas, de modo que se habrá de calcular la posición de cada punto para cada fotograma. Eso se traduce en 864 millones de cálculos por segundo, y es probable que cada cálculo sea bastante complejo, implicando docenas o centenares de instrucciones en código máquina. Esto añade hasta billones de instrucciones por segundo. El recuadro *Pruebas de tiempo* muestra una comparación aproximada del tiempo que consume producir una secuencia de película de 10 minutos en un superordenador y en un micro basado en el Z80.



Los superordenadores, tal como tiende a llamarse a los mayores ordenadores centrales, trabajan de forma muy parecida a los microordenadores; las instrucciones y los datos se traen de la memoria, el procesador manipula los datos según sus instrucciones y los resultados se almacenan en la memoria. La principal diferencia radica en la escala y la velocidad a la cual se llevan a cabo estas operaciones, y en la forma en que se construyen los componentes de los ordenadores.

Ilustraremos un superordenador «típico» refiriéndonos al Cray-1 S/4400, fabricado por Cray Research. El primer Cray-1 se instaló en 1976, sólo cuatro años después de la creación de la compañía. Desde entonces se ha establecido como el superordenador más famoso (y uno de los más populares).

La CPU del Cray-1 ocupa una cabina semicircular de 6,5 pies (1,98 m) de altura, de forma similar al de un sofá curvo (debajo de los «asientos» está el sistema de refrigeración y las fuentes de alimentación). Obtiene su velocidad mediante el empleo de lógica de semiconductores bipolares (los semiconductores bipolares son transistores «normales», al contrario que los MOS, CMOS, NMOS, FET —*field effect transistor*— y otros tipos de semiconductores). La lógica bipolar y la memoria vienen en más de 200 000 circuitos integrados en 3 400 placas de circuito impreso separadas, para conectar las cuales se utilizan más de 90 kilómetros de cable.

El tamaño de palabra que se utiliza para los cálculos es de 64 bits de ancho (la cantidad de datos que se procesan a la vez es ocho veces más que en un Z80 de ocho bits) y la velocidad del reloj del sistema es de 80 MHz (80 millones de ciclos por segundo). Una suma típica de 64 bits lleva sólo 37,5 nanosegundos. Una suma de ocho bits en un Z80 operando a 4 MHz (p. ej., `ADD A,n`) lleva 1,75 microsegundos.

La memoria principal del Cray es una parte inte-



gral de la CPU. Contiene 32 megabytes organizados en 4 194 304 palabras. La memoria puede transferir hasta 2 560 millones de bytes por segundo.

Como es de suponer, el conjunto de registros de la CPU del Cray es impresionante. Hay 72 registros de dirección (cada uno de 24 bits de ancho), 72 registros escalares (cada uno de 64 bits de ancho) y ocho registros de vector (cada uno de 64 palabras de ancho). El espacio total de registros de la CPU del Cray es, por lo tanto, de 4 888 bytes (el del microprocesador Z80 es de apenas 26).

A diferencia de la mayoría de los otros ordenadores, que son unidades independientes, los ordenadores Cray están diseñados para usarse como extensiones de miniordenadores u ordenadores centrales ya existentes. El ordenador frontal (*front end*) actúa como una interface, tomando la entrada de los terminales o lectores de tarjetas, y alimentando la salida a periféricos tales como impresoras o unidades de almacenamiento en cinta magnética.

Entre la CPU del Cray y el ordenador central frontal se halla el subsistema de E/S Cray, diseñado para facilitar las inmensas demandas de uso de la CPU, y una interface frontal, diseñada para adecuar el sistema Cray a las características específicas del IBM, DEC, Data General u otros ordenadores frontales. El subsistema de E/S Cray consta de entre dos y cuatro procesadores de E/S, cada uno de los cuales es un potente miniordenador en toda la regla.

Esta descripción de la arquitectura del Cray-1 apenas si toca la superficie, aunque el diagrama de bloques de la CPU tal vez lo ayude a hacerse una idea de su nivel de sofisticación. No hay espacio suficiente para hablar del juego de instrucciones, las 13 unidades funcionales, como se las llama, que pueden operar en paralelo, ni del proceso de vectores que permite operar, mediante una única instrucción, sobre hasta 64 pares de operandos. Basta con decir que el Cray es una máquina *muy* potente.

Aplicaciones específicas

Pronóstico meteorológico

Los pronósticos meteorológicos son ahora el resultado de la recogida de datos meteorológicos internacionales, comunicación por satélite y modelación por ordenador. Un «modelo» por ordenador del tiempo en el mundo implica cientos de millones de cálculos en vastas matrices de datos, necesitándose las predicciones en cuestión de horas. Sólo ordenadores inmensamente potentes como el Cray pueden manipular esta cantidad de datos con suficiente rapidez

Dinámica de fluidos

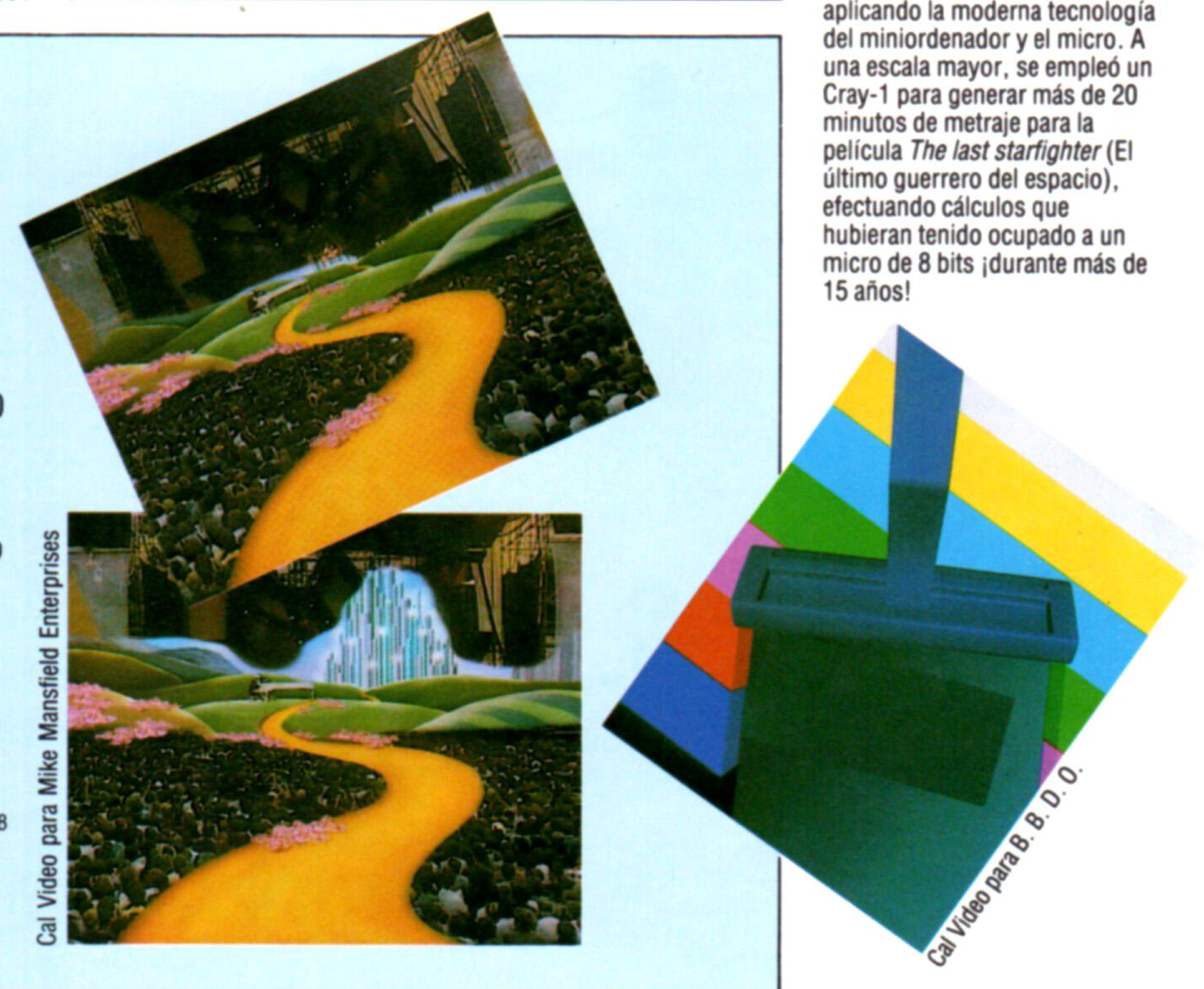
La dinámica de fluidos juega un importante papel en el diseño de coches eficaces desde el punto de vista del combustible, los sistemas refrigerantes de las centrales de energía nuclear, la ingeniería aeronáutica y muchas otras aplicaciones. El análisis y la predicción del movimiento de los fluidos exige analizar montones de datos. Los problemas se agravan si se necesitan resultados en tiempo real. La cuestión estriba en que cada partícula de un fluido (y hay incontables trillones de ellas) tendrá un efecto sobre todas las otras partículas del sistema cuando éste se mueva. Por consiguiente, calcular el comportamiento de un sistema completo de fluidos exige un poder informático masivo

Previsión económica

Construir modelos de economía es notablemente difícil. Al igual que en la dinámica de fluidos, un pequeño cambio en cualquier parte del sistema puede tener efectos que se extiendan a través del resto del sistema. Intentar crear un modelo económico de todo el mundo es apenas viable, pero aun los modelos simplificados poseen una complejidad abrumadora. Se necesitan ordenadores rápidos y muy veloces para que los analistas no hayan de esperar durante meses los resultados

Pruebas de tiempo

Para comparar el poder de proceso de un superordenador Cray-1 y un ordenador personal basado en el Z80, como el Amstrad CPC-464, consideremos los problemas que entraña hacer una secuencia de película de 10 minutos de gráficos animados y en alta resolución. Supongamos que se requiere una resolución de imagen de 6 000 por 6 000 pixels y 24 fotogramas por segundo. Supongamos, asimismo, que es necesario calcular individualmente cada pixel de cada nuevo fotograma, y que para la versión Z80 se precisarán 100 instrucciones en código máquina, con un tiempo de ejecución medio de 19 ciclos de reloj (4,75 microsegundos). Para el Cray-1, con sus poderosas instrucciones de proceso de vectores, diremos que se necesitan 25 instrucciones en código máquina (para pecar de conservadores) con un tiempo de ejecución promedio de cuatro ciclos de reloj (50 nanosegundos). Un Z80 tardaría $6000^2 \times 24 \times 60 \times 10 \times 100 \times 4,75 \times 10^{-6} = 2,4624 \times 10^8$ segundos, o 7,8 años. Un Cray-1 tardaría $6000^2 \times 24 \times 60 \times 10 \times 25 \times 50 \times 10^{-9} = 6,48 \times 10^5$ segundos, o 7,5 días



Cal Video para Mike Mansfield Enterprises

Cal Video para B. B. D. O.

Revolución visual
Desde el *Goodbye yellow brick road* (Adiós, camino de ladrillos amarillos), de Elton John, hasta los anuncios de Gillette, los gráficos generados por ordenador están revolucionando la producción de videos y películas. Se pueden crear fácilmente imágenes como éstas aplicando la moderna tecnología del miniordenador y el micro. A una escala mayor, se empleó un Cray-1 para generar más de 20 minutos de metraje para la película *The last starfighter* (El último guerrero del espacio), efectuando cálculos que hubieran tenido ocupado a un micro de 8 bits ¡durante más de 15 años!

¡No pases de largo!

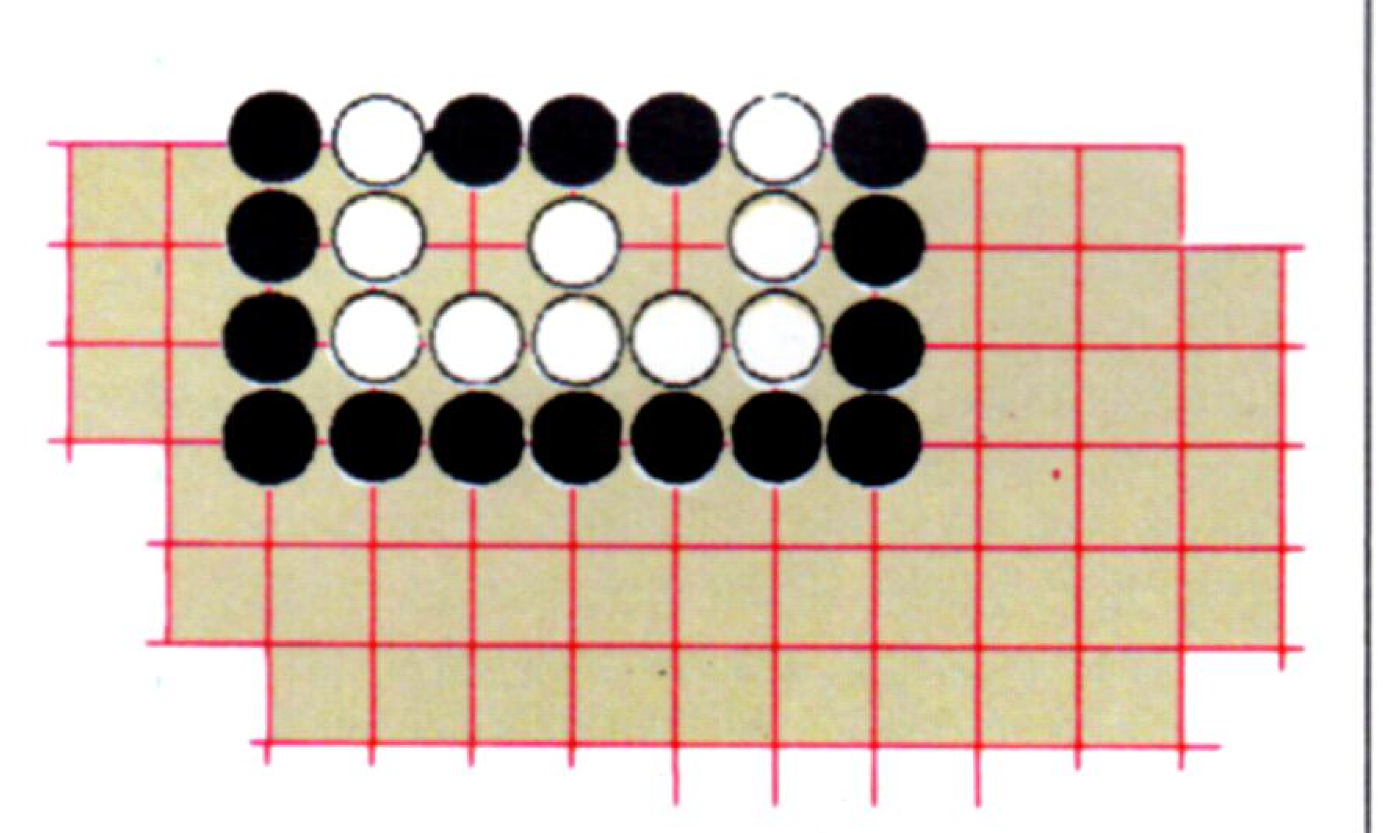
Prosiguiendo con nuestro proyecto de programación, listaremos las rutinas de entrada/salida para el Commodore 64, el Spectrum y el Amstrad CPC 464/664

Estas rutinas de entrada/salida tienen por finalidad crear la visualización del tablero para el juego; proporcionan, además, subrutinas de propósito general para imprimir mensajes y aceptar entradas.

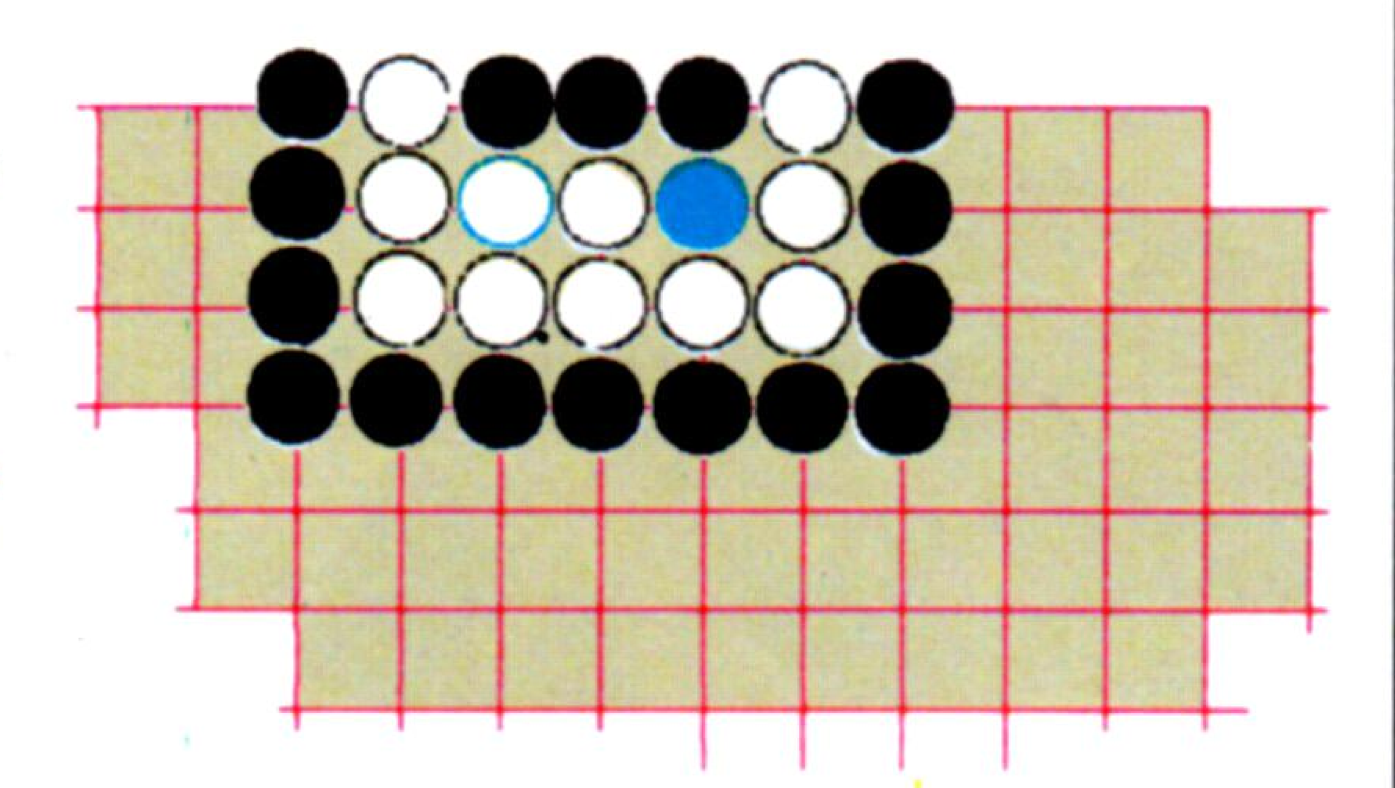
En la medida de lo posible, los listados que ofrecemos constituyen el segmento de programa equivalente al que ofrecíamos para el BBC Micro, y gran parte de la descripción ofrecida para los listados del capítulo anterior es igualmente válida para las versiones del *go* para el Commodore 64, el Sinclair Spectrum y el Amstrad CPC 464/664. Para mantenernos en un nivel de sencillez, en todas las versiones del programa se utilizan los mismos números de línea; los algoritmos originales sólo se alteran allí donde las diferencias en cuanto a arquitectura de la máquina y versión de BASIC se vuelven cruciales. Las principales diferencias entre la versión para el BBC Micro y las que ofrecemos aquí, aparte de la forma en que las cuatro máquinas manipulan sus visualizaciones en pantalla, es que las versiones de BASIC del Amstrad, el Spectrum y el Commodore 64 carecen de procedimientos y funciones multilíneas. No obstante, hemos simulado estos últimos mediante el empleo de subrutinas.

SEKI

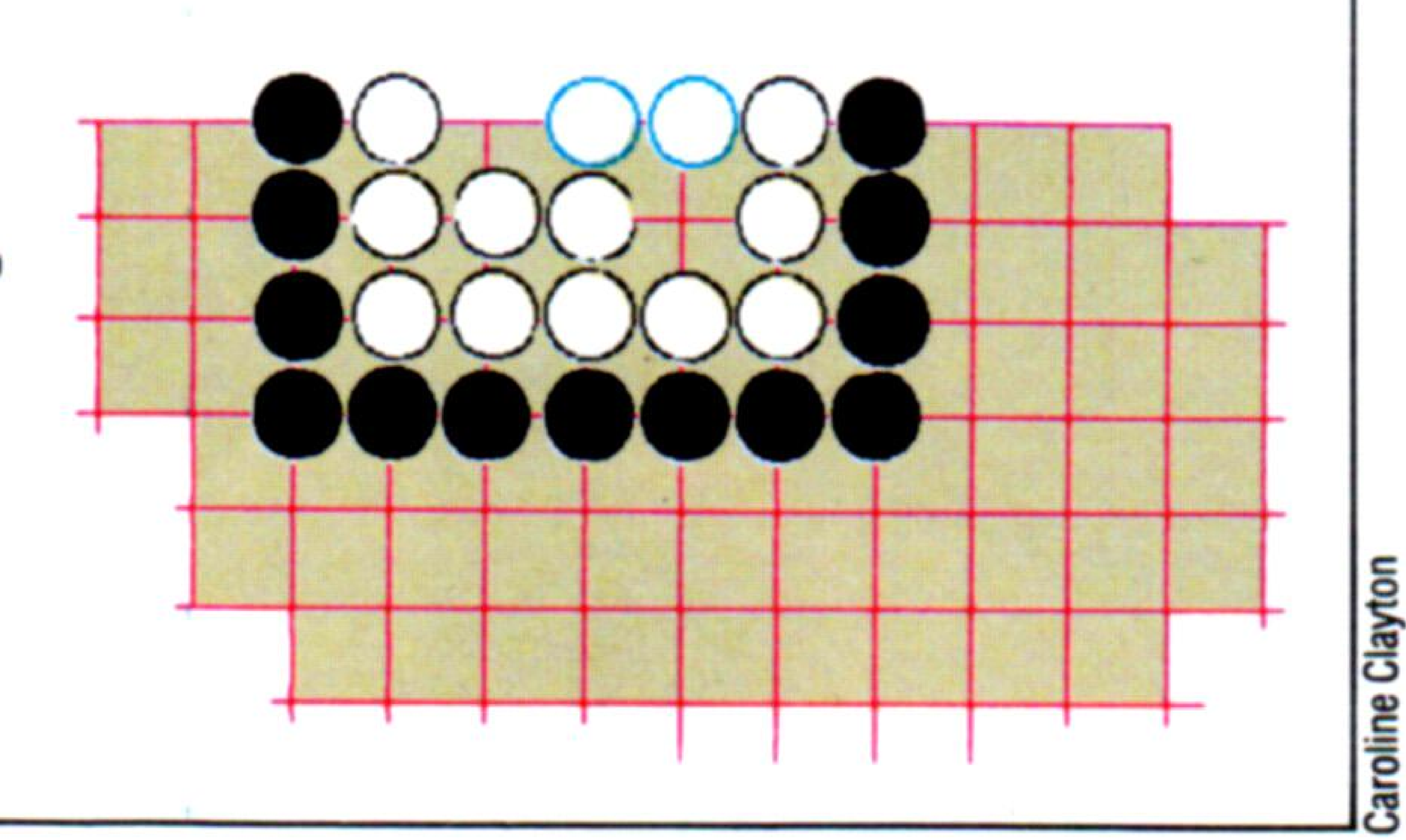
¿Quién tiene rodeado a quién?
El grupo blanco está completamente rodeado por el grupo negro, más numeroso, y sólo le quedan dos licencias. Sin embargo, este grupo también tiene rodeado a un grupo negro más pequeño de tres fichas



Error fatal
Si las negras juegan en una de las dos licencias restantes de las blancas, en un intento por capturar el grupo blanco, entonces las blancas pueden jugar en la licencia restante. Puesto que estas dos licencias las comparten las blancas y el grupo negro más pequeño, entonces se podrá suprimir del tablero este grupo negro



Seguridad
Ahora el grupo blanco se puede poner a salvo formando dos ojos, jugando en una de las dos posiciones que se indican. Las negras habrán dejado vacantes las dos licencias compartidas originales, situación que en *go* se denomina SEKI



Caroline Clayton

Módulo Uno

Commodore 64:

```
10 POKE 56578,PEEK(56578)OR 3:POKE
   56576,(PEEK(56576)AND 252)OR 1
15 POKE 648,132:POKE 52,128:POKE 56,128
20 DIM SK%(300)
30 POKE 53280,0:POKE
   53281,0:PRINT"[BORR.][ABAJ][ABAJ][ABAJ][ABAJ]
   [ABAJ][ABAJ][ABAJ][ABAJ][ABAJ][ABAJ]          [BLANC]
   POR FAVOR ESPERE":GOSUB 170
40 GOSUB 1270
50 PRINT"[BORR.]":GOSUB 1730
60 MOVIMIENTO%=MOVIMIENTO%+1:REM AQUI
   MOVIMIENTOS BLANCAS
70 IF FIN% GOTO 100
80 MOVIMIENTO%=MOVIMIENTO%+1:REM AQUI
   MOVIMIENTOS NEGRAS
90 IF NOT FIN% GOTO 60
100 IP%=22:IM%=9:IW%=1:GOSUB 1990
110 IF A$="S" GOTO 40
115 IF A$<>"N" GOTO100
120 MP%=24:MM%=5:O$="":GOSUB 2160
130 GOTO 130
170 REM RUTINA INICIALIZAR
190 NEGRAS%=1:BLANCAS%=2:COLOR%=3
200 MARCADOR%=4:LICENCIA%=8
220 TABLERO=49152
240 DIM CAPTURA%(2)
250 EJE$="[C 3] ABCDEFGHIJKLMNO"
260 DIM DIR%(4)
270 FOR L=1 TO 4
280 READ DIR%(L)
290 NEXT
300 DATA 16,1,-16,-1
310 GOSUB 390
350 RETURN
390 REM RUTINA LEER MENSAJES
420 DIM MENS$(9)
430 FOR M=0 TO 9
440 READ MENS$(M)
450 NEXT
460 DATA "O.K. ESTOY PENSANDO..."
470 DATA "ENTRADA ILEGAL: "
480 DATA "FICHA YA EN ESE LUGAR: "
490 DATA "ILEGAL. KO EN ESE LUGAR: "
500 DATA "SUICIDIO ILEGAL EN ESE LUGAR: "
510 DATA "O.K. JUEGO TERMINADO. PULSE
   RUN/STOP"
520 DATA ""
```



```
530 DATA "CUANTAS FICHAS DE HANDICAP
    PUEDO              TENER (2-9)?"
540 DATA "DIGITE SU MOVIMIENTO (P.EJ. H8), PASO
    0             ABANDONO:"
550 DATA "QUIERE JUGAR OTRA VEZ (S/N)?"
560 RETURN
1270 REM RUTINA DE INTRODUCCION
1280 GOSUB 1340
1290 GOSUB 1450
1340 REM RUTINA INIC-JUEGO
1345 PILA%=1
1360 A1$="          ":A2$="          "
1370 POSIC%=0:MOVIMIENTO%=1
1380 FIN%=0
1390 CAPTURA%(1)=0:CAPTURA%(2)=0
1410 RETURN
1450 REM RUTINA PANTALLA-TITULOS
1460 POKE 53280,0:POKE 53281,0
1470 PRINT
     "[CLEAR][DOWN][DOWN][DOWN][DOWN]-
                 [c 3][RVSON][c V][c I][c
     C][RVSOFF]   [RVSON][c V][c I][c C]"
1480 PRINT"                 [RVSON]   [RVSOFF][c D][c
     I]  [RVSON]   [RVSOFF]  [RVSON]   "
1490 PRINT"                 [RVSON]   [RVSOFF]   [RVSON]-
       [RVSOFF]   [RVSON]   [RVSOFF]   [RVSON]   "
1550 PRINT"                 [c C][RVSON][c I][RVSOFF][c
     V]   [c C][RVSON][c I][RVSOFF][RVSOFF][c F]"
1510 PRINT"[DOWN][DOWN]              [CYAN] POR MARCUS
     JEFFERY"
1520 PRINT"[DOWN][PURPLE]TU JUGARAS CON LAS FICHAS
     [WHITE]BLANCAS[PURPLE], y"
1530 PRINT"EL ORDENADOR (AL SER MAS DEBIL!)
     JUGARA"
1540 PRINT"CON LAS FICHAS [c 3]ROJAS[PURPLE]CON UN
     HANDICAP"
1550 PRINT"                   DE VENTAJA."
1560 IP%=20:IM%=7:IW%=1:GOSUB 1990
1565 HND%=VAL(A$)
1570 IF HND%<2 OR HND%>9 GOTO 1560
1590 RETURN
1730 REM RUTINA IMPRESION-TABLERO
1750 PRINT"[HOME][YELLOW]                FICHAS
     CAPTURADAS POR:";
1760 PRINT"[WHITE]          MOVIMIENTO "
1770 PRINT"[YELLOW]           BLANC=[CYAN]
     ";CAPTURA%(2);
1780 PRINT TAB(18);"[YELLOW] NEGRAS=[CYAN]
     ";CAPTURA%(1);
1790 PRINT TAB(31);"[WHITE]";MOVIMIENTO%
1880 PRINT TAB(12);"[DOWN]";EJE$
1810 FOR Y=15 TO 1 STEP -1
1820 PRINT TAB(9);"[c 3]";RIGHT$("  "+STR$(Y),2);"  ";
1830 FOR X=1 TO 15
1840 PP%=PEEK(TABLERO+16*Y+X)
1850 IF PP%=1 THEN PRINT"[c 3][s Q]";:GOTO 1880
1860 IF PP%=2 THEN PRINT"[WHITE][s Q]";:GOTO
     1880
1870 PRINT"[c 6][s +]";
1880 NEXT
1890 PRINT"[c 3]";RIGHT$("  "+STR$(Y),2)
1900 NEXT
1910 PRINT TAB(12);EJE$
1920 PRINT"[DOWN][YELLOW]          ULT. MOV.";
1930 ITCP%=POSIC%:GOSUB 2260:PRINT TAB(24);
     A1$
1940 PRINT TAB(18);"[DOWN][CYAN]";A2$
1950 RETURN
1990 REM RUTINA DE ENTRADA
2000 IS%=0
2010 POLE 780,0:POKE 781,IP%+1:POKE 782,39:SYS
     65520
2015 FOR II=1 TO 79:PRINT CHR$(20);:NEXT
2020 A$=""
2030 PRINT"[WHITE]        ";MENS$(IM%);"  ";
2060 GET I$:IF I$="" GOTO 2060
2065 IF ASC(I$)=13 GOTO 2120
2070 IF ASC(I$)<>20 GOTO 2100
2080 IF IS%>0 THEN IS%=IS%-1:A$=LEFT$(A$,IS%):
     PRINT I$;:GOTO 2060
2085 GOTO 2060
2100 IF IS%<IW% THEN IS%=IS%+1:A$=A$+I$:PRINT
     "[CYAN]";I$;
2110 GOTO 2060
2120 RETURN
2160 REM RUTINA DE MENSAJES
2190 POKE 780,0:POKE 781,MP%:POKE 782,39:SYS
     65520
2195 FOR ML=1 TO 39:PRINT CHR$(20);:NEXT
2200 PRINT"       [RVSON][c
     3]";MENS$(MM%);O$;"[RVSOFF]";
2220 RETURN
2260 REM RUTINA INT-TO-CHAR
2270 IF ITCP%=0 THEN PRINT"[CYAN] HANDICAP";:
     RETURN
2275 C$=CHR$(ITCP%-16*INT(ITCP%/16)+64)+MID$
     (STR$(INT(ITCP%/16)),2)
2280 PRINT"[CYAN]";C$;"        ";:RETURN
```

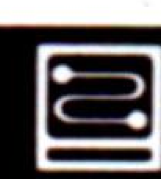
Amstrad CPC 464/664:

```
10 MEMORY &9FFF
20 DIM s(300):REM pila
30 GOSUB 170: REM inic
40 GOSUB 1270:REM introduccion
50 CLS:GOSUB 1730:REM imprimir
   tablero
60 mov%=mov%+1:REM aqui movimientos
   blancas
70 IF mas% THEN 60
80 mov%=mov%+1:REM aqui movimientos
   negras
90 IF NOT mas% THEN 60
100 ip%=23: im%=9: iw%=1:GOSUB
    1990
110 IF a$="S" THEN 40
115 IF a$<>"N" THEN 100
120 mp%=25:mm%=5:o$="":GOSUB
    2160:REM mensaje
130 END
170 REM rutina inic
190 negras%=1:blancas%=2:color%=3
195 INK 1,26:REM blanco
197 INK 2,24:REM amarillo
200 marcador%=4:licencia%=8
220 tablero=&A000
240 DIM captura%(2)
250 eje$="A B C D E F G H I J K L M N O"
260 GOSUB 390:REM leer mensajes
290 DIM dir%(4)
300 RESTORE 340
310 FOR I%=1 TO 4
320 READ dir%(I%)
330 NEXT I%
340 DATA 16,1,-16,-1
350 RETURN
390 REM rutina leer mensajes
410 RESTORE 460
420 DIM mens$(9)
430 FOR m%=0 TO 9
440 READ mens$(m%)
450 NEXT m%
460 DATA "O.K. ESTOY PENSANDO..."
470 DATA "Entrada ilegal: "
480 DATA "Ficha ya en ese lugar: "
490 DATA "Ilegal. Ko en ese lugar:"
500 DATA "Ilegal. Suicidio en ese lugar: "
510 DATA "      O.K. JUEGO
    TERMINADO"
520 DATA ""
530 DATA "Cuantas fichas puedo tener
    de        handicap (2-9)?"
540 DATA "Digita tu movimiento (p.ej. H8) PASO
    o ABANDONO:"
550 DATA "Quieres jugar otra vez (S/N)?"
560 RETURN
1270 REM rutina de presentacion
1280 GOSUB 1340:REM inic juego
1290 GOSUB 1450:REM pantalla titulos
1300 RETURN
1340 REM rutina inic juego
1345 pila%=1
1360 atari1$="          ":atari2$=""
1370 posicion%=:mov%=1
1380 mas%=0
1390 captura%(1)=0:captura%(2)=0
1410 RETURN
1450 REM rutina pantalla titulos
1460 CLS
1462 FOR c=1 TO 20 STEP 3
1465 RESTORE 1474
1470 MOVE 260+c,320+c
1472 FOR i=1 TO 8:READ x,y:DRAWR x,y,3:
     NEXTi
1474 DATA -10,10,-50,0,-10,-10,0,-80,10,
     -10,50,0,10,10,0,20
1476 MOVER -10,0:DRAWR 20,0
1478 MOVE 380+c,320+c
1480 FOR i=1 TO 8: READ x,y=DRAWR x,y:
     NEXTi
1482 DATA -10,10,-50,0,-10,-10,0,-80,10,
     -10,50,0,10,10,0,80
1484 NEXT c
1500 LOCATE 12,13:PEN 3:PRINT
1510 LOCATE 1,16:PEN 2:PRINT"Jugaras con las
     fichas";
1520 PEN 1:PRINT"blancas";:PEN 2
1530 PRINT"y el ordenador (al ser mas debil!)
     jugara"
1540 PRINT"con las";:PEN 3:PRINT"rojas";:PEN
     2:PRINT"con una ventaja de"
```

```
1550 PRINT TAB(13)"handicap"
1560 ip%=22:im%=7:iw%=1:GOSUB 1990:REM
     entrada
1565 hand%=ASC(a$)-48
1570 IF hand%<2 OR hand%>9 THEN
     1560
1590 RETURN
1730 REM rutina impresion tablero
1735 INK 0,4:BORDER 9:REM magenta/
     verde
1750 LOCATE 3,1:PEN 2:PRINT" Fichas capturadas
     por:";TAB(32)"Movimiento"
1770 PRINT TAB(3)"Blancas =";:PEN 3:PRINT
     captura%(2);
1780 PEN 2:PRINT "Negras =";:PEN 3:PRINT
     captura%(1);
1790 PRINT TAB(33)mov%
1800 PRINT:PRINT TAB(5)eje$
1810 FOR y%=15 TO 1 STEP -1
1820 LOCATE 2,20,-y%:PRINT
     RIGHT$(""+STR$(y%),2);
1830 FOR x%=1 TO 15
1835 PRINT "";
1840 p%=PEEK(tablero+16*y%+x%)
1850 IF p%=1 THEN PEN 3:PRINT"O";:GOTO
     1880
1860 IF p%=2 THEN PEN 1:PRINT"O";:GOTO
     1880
1870 PEN 2:PRINT"+";
1880 NEXT x%
1890 PEN 3:PRINT TAB (34)y%
1900 NEXT y%
1910 PRINT TAB(5)eje$
1920 PEN 2:PRINT TAB(3)"Mi ultimo movimiento
     fue:";
1930 itcp%=posicion%:GOSUB 2260:PEN
     1:PRINT TAB(30) atari1$
     1940 LOCATE 16,23:PRINT atari2$
1950 RETURN
1990 REM rutina de entrada
2000 is%=0
2010 LOCATE 3,ip%:PRINT SPACE$(62)
2020 a$=""
     LOCATE 3,ip%:PEN 3:PRINT
     mens$(im%);
2060 i$="":WHILE i$="":i$=INKEY$:
     WEND
2065 IF i$=CHR$(13) THEN 2120
2070 IF i$<>CHR$(127) THEN
     2090
2080 IF is$>0 THEN is%=is%-1:a$=
     LEFT$ (a$,is%):PRINT CHR$(8);"";
     CHR$(8);
2085 GOTO 2060
2090 IF i$>="a" AND i$<="z" THEN
     i$=CHR$(ASC(i$)-32)
2100 IF is%<iw% THEN is%=is%+1:a$=
     a$+i$:PEN 2:PRINT i$;
2110 GOTO 2060
2120 RETURN
2160 REM rutina de mensajes
2190 LOCATE 3,mp%:PRINT SPACE$
     (36);
2200 LOCATE 3,mp%:PEN 3:PRINT mens$(mm%);
     o$
2220 RETURN
2260 REM rutina int-to-char
2270 IF itcp%=0 THEN PEN 1:PRINT"Handicap"
     ;:RETURN
2275 c$=CHR$(itcp%-16*INT(itcp%/16)+64)
     :r$=MID$(STR$(INT(itcp%/16)),2)
2280 PEN 1:PRINT c$;r$;:RETURN
```

Sinclair Spectrum

```
  10 CLEAR 63999
  20 DIM s(300)
  30 PRINT AT 10,10;"POR FAVOR ESPERA":GO
     SUB 170
  40 GO SUB 1270
  50 CLS:GO SUB 1730
  60 LET movimiento=movimiento+1:REM aqui
     movimientos blancas
  70 IF fin THEN GO TO 100
  80 LET movimiento=movimiento+1:REM aqui
     movimientos negras
  90 IF NOT fin THEN GO TO 60
 100 LET ip=20: LET im=10: LET iw=1: GO SUB
     1990
 110 IF a$="S" THEN GOTO 40
 115 IF a$<> "N" THEN GO TO 100
 120 LET mp=21: LET mm=6: LET o$="": GO
     SUB 2160
 130 STOP
 170 REM rutina inicializar
 190 LET negras=1: LET blancas=2: LET
     color=3
 200 LET marcador=4: LET licencia
     =8
 220 LET tablero=64000
 240 DIM c(2)
 260 GO SUB 390
 290 DIM d(4)
 300 RESTORE 340
 310 FOR l=1 TO 4
 320 READ d(l)
 330 NEXT l
 340 DATA 16,1,-16,-1
 350 RETURN
 390 REM rutina leer-mensajes
 410 RESTORE 460
 420 DIM m$(10,43)
 430 FOR m=1 TO 10
 440 READ m$(m)
 450 NEXT m
 460 DATA "O.K. ESTOY PENSANDO..."
 470 DATA "Entrada ilegal: "
 480 DATA "Ficha ya en ese lugar: " .
 490 DATA "Ilegal. Ko en ese lugar: "
 500 DATA "Ilegal. Suicidio en ese
     lugar:"  .
 510 DATA "      O.K. JUEGO
     TERMINADO."
 520 DATA ""
 530 DATA "Cuantas fichas puedo tener de
     handicap (2-9)?"
 540 DATA "Digita tu movimiento (p.eje. H8),
     PASO o ABANDONO :"
 550 DATA "Quieres jugar otra vez       (S/N)?"
 560 RETURN
1260 :
1270 REM rutina de presentacion
1280 GO SUB 1340
1290 GO SUB 1450
1300 RETURN
1340 REM rutina inic-juego
1345 LET pila=1
1360 LET x$="        ": LET y$="        "
1370 LET posicion=0: LET movimiento=1
1380 LET fin=0
1390 LET c(1)=0: LET c(2)=0
1410 RETURN
1450 REM rutina pantalla titulos
1460 BORDER 0: PAPER 0: CLS
1470 INK 2: PRINT AT 3,12; "sh23sh1
     sh23sh1"
1480 PRINT AT 4,12;"sh84sh3 sh8 sh8"
1490 PRINT AT 5,12;"sh8 sh8     sh8 sh8"
1500 PRINT AT 6,12;"132     132"
1510 PRINT AT 8,7; INK 5;"por Marcus
     Jeffery"
1520 PRINT:PRINT INK 3:"Tu jugaras con las
     fichas"; INK 7:"blancas"; INK 3
1530 PRINT INK 3;"y el ordenador (por ser mas
     debil)"
1540 PRINT INK 3;"jugara con las fichas";INK
     2;"rojas    " ;INK 3;"con"
1550 PRINT INK 3;"       una ventaja de
     handicap."
1560 LET ip=15: LET im=8: LET iw=1: GO SUB
     1990
1565 LET hand=CODE (a$)-48
1570 IF hand<2 OR hand>9 THEN GO TO
     1560
1590 RETURN
1730 REM rutina impresion tablero
1750 PRINT AT 0,0; INK 6;" Fichas capturadas
     por:";
1760 PRINT INK 7;"        Mov."
1770 PRINT INK 6;" Blancas=
     "; INK 5;c(2);
1780 PRINT TAB 10; INK 6;"Negras=
     "; INK 5;c(1);
1790 PRINT TAB 27; INK 7; movimiento
1810 FOR y=15 TO 1 STEP -1
1820 PRINT AT 17-y,5; INK 2;(""+STR$ y)(LEN
     STR$ y TO);" ";
1830 FOR x=1 TO 15
1840 LET pp=PEEK (tablero+16*y+x)
1850 IF pp=1 THEN PRINT PAPER 4; INK
     2;"O";:GO TO 1880
1860 IF pp=2 THEN PRINT PAPER 4; INK
     7;"O";:GO TO 1880
1870 PRINT PAPER 4; INK 0;"+";
1880 NEXT x
1890 PRINT INK 2;" ";y
1900 NEXT y
1910 PRINT TAB 8; INK 2;"ABCDEFGHIJKLMNO"
1920 PRINT INK 6;"Ultimo movimiento: ";
1930 LET itcp=posicion: GO SUB 2260: PRINT
     TAB 20: INK 5;x$
1940 PRINT AT 20,20; INK 5;y$:
     BEEP 1,0
1950 RETURN
1990 REM rutina de entrada
2000 LET is=0
2010 PRINT AT ip,0;:FOR i=1 TO 62: PRINT "
     ";:NEXT i
2020 LET a$=" "
2030 PRINT AT ip,0; INK 7; m$(im);
2060 LET i$=INKEY$: IF i$=" " THEN GO TO
     2060
2065 IF CODE i$=13 THEN GO TO
     2120
2070 IF CODE i$<>12 THEN GO TO
     2090
2080 IF is>0 THEN LET is=is-1:LET a$=a$(1 TO
     LEN a$-1): PRINT CHR$ (8);"
     ";CHR$(8);:GO TO 2060
2085 GO TO 2060
2090 IF i$>="a" AND i$<="z" THEN LET
     i$=CHR$ (CODE i$-32)
2100 IF is<iw THEN LET is=is+1: LET
     a$=a$+i$: PRINT INK5; i$;
2110 GO TO 2060
2120 RETURN
2160 REM rutina de mensajes
2190 PRINT AT mp,0;:FOR m=1 TO 30: PRINT "
     ";: NEXT m
2200 PRINT AT mp,0;  INK 2;m$(mm, TO 26);
     o$;
2220 RETURN
2260 REM rutina int-to-char
2270 IF itcp=0 THEN PRINT INK
     5;"Handicap";:RETURN
2275 LET c$=CHR$ (itcp-16*INT (itcp/16)+64):
     LET r$=STR$ (INT (itcp/16))
2280 PRINT INK 5;c$;r$;:RETURN
```



Datos básicos (II)

Proseguiremos nuestro análisis del mapa de memoria del C64, con información facilitada por Commodore Business Machines

ETIQUETA	DIRECCIÓN HEXA	POSICIÓN DECIMAL	DESCRIPCIÓN
LASTPT	0017-0018	23-24	Última dirección de serie temporal
TEMPST	0019-0021	25-33	Pila para series temporales
INDEX	0022-0025	34-37	Área punteros utilidades
RESHO	0026-002A	38-42	Producto de multiplicar coma flotante
TXTTAB	002B-002C	43-44	Puntero: inicio texto en BASIC
VARTAB	002D-002E	45-46	Puntero: inicio variables BASIC
ARYTAB	002F-0030	47-48	Puntero: inicio tablas BASIC
STREND	0031-0032	49-50	Puntero: fin tablas BASIC (+ 1)
FRETOP	0033-0034	51-52	Puntero: extremo inferior almacenamiento series
FRESPC	0035-0036	53-54	Puntero series utilidades
MEMSIZ	0037-0038	55-56	Puntero: dirección más alta empleada por BASIC
CURLIN	0039-003A	57-58	Número línea actual BASIC
OLDLIN	003B-003C	59-60	Número línea anterior BASIC
OLDTXT	003D-003E	61-62	Puntero: sentencia BASIC de CONT
DATLIN	003F-0040	63-64	Número actual línea DATA
DATPTR	0041-0042	65-66	Puntero: dirección ítem actual DATA
INPPTR	0043-0044	67-68	Vector: rutina INPUT
VARNAM	0045-0046	69-70	Nombre actual variable BASIC
VARPNT	0047-0048	71-72	Puntero: datos variables en curso de BASIC
FORPNT	0049-004A	73-74	Puntero: variable índice de FOR/NEXT
	004B-0060	75-96	Puntero temp/Área datos
FACEXP	0061	97	Acumulador # 1 coma flotante: exponente

Ratón mágico

El Magic Mouse, de SMC Supplies, es el primer dispositivo de ratón creado para el Commodore 64

A la vista de la actual popularidad de que gozan los sistemas operativos basados en ratón, quizá resulte difícil creer que inicialmente se produjera tanta controversia dentro de la industria del ordenador en cuanto a si el sistema basado en ratón era o no aceptable, por no hablar de deseable. El fracaso comparativo del Apple Lisa, el primer micro que tuvo un sistema operativo basado en ratón, pareció confirmar la idea de que los usuarios no querían estos sistemas.

El posterior éxito del Macintosh silenció, sin embargo, las críticas, y justificó la fe que tenía depositada Apple en su sistema. Las actitudes han cambiado drásticamente; los fabricantes de ordenadores se están apresurando a producir máquinas basadas en ratón, y muchas empresas independientes están editando paquetes de ratón para máquinas existentes.

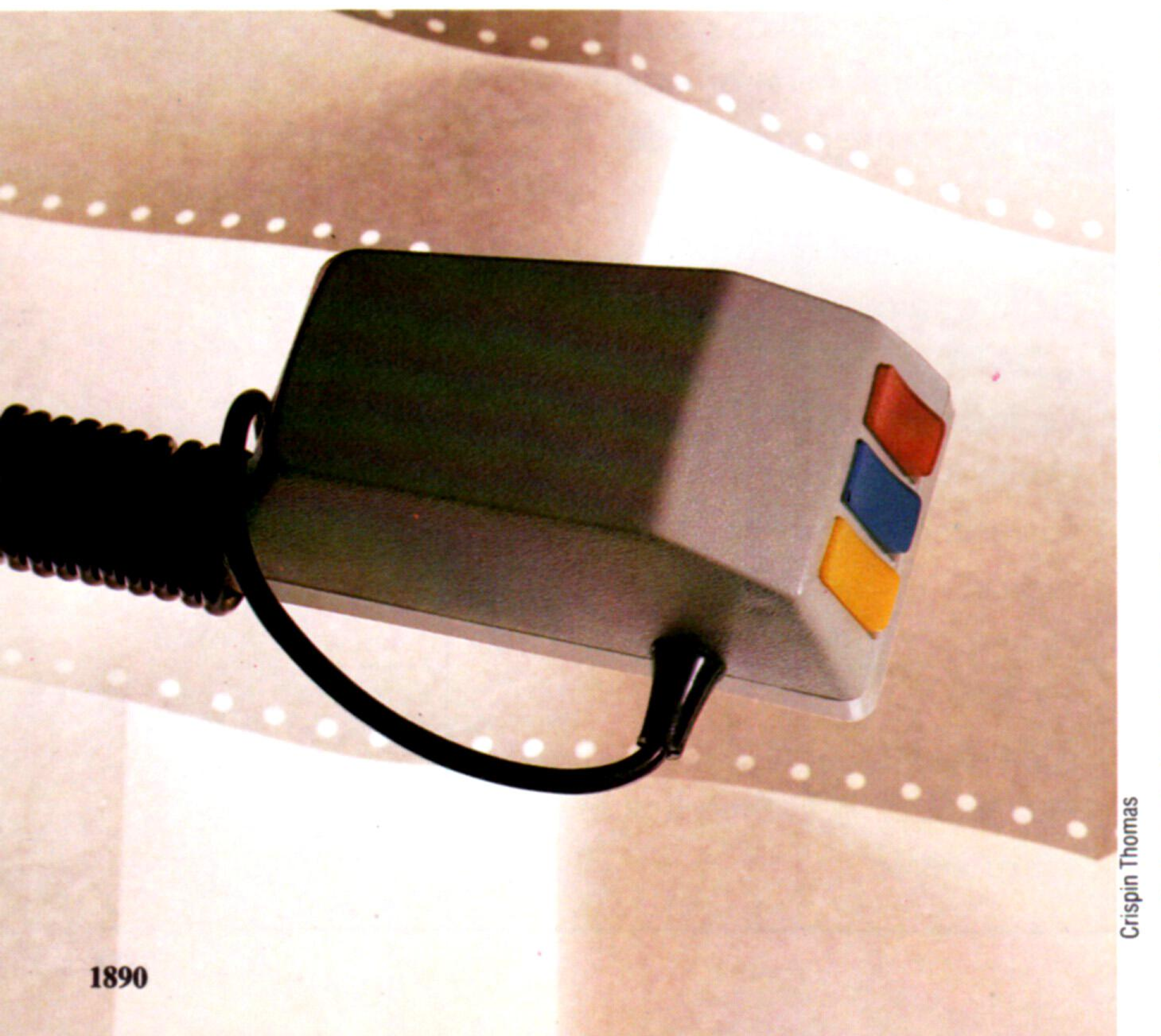
El Magic Mouse (ratón mágico) de SMC Supplies es uno de los primeros de tales productos que ha salido para el Commodore 64. Con el ratón se incluye el software, tanto en cassette como en disco, que contiene cuatro paquetes de aplicaciones. El ratón SMC es más grande que la mayoría de los dispositivos similares, y con sus 125 por 66 por 50 mm, su tamaño es casi el doble que el del ratón Apple.

El Magic Mouse también tiene tres botones de colores en la parte delantera, frente a los dos que suele haber normalmente. Los botones se emplean para seleccionar funciones que, por supuesto, varían a tenor de la aplicación que se esté utilizando. Por dentro hay una bola de plástico duro que hace presión contra un par de codificadores de eje.

Aparentemente, el diseño original llevaba una conexión de bola metálica estándar, pero se descubrió que la misma no era adecuada y se retrasó el lanzamiento del dispositivo para permitir la incorporación de los recambios de goma. No obstante, el ratón parece funcionar a la perfección.



Eficaz herramienta
El SMC Magic Mouse es uno de los primeros sistemas basados en ratón que se desarrolla para el Commodore 64. A pesar de no estar diseñado como un auténtico paquete WIMP *(windows, icons, mice, programming:* ventanas, iconos, ratones, programación), se puede emplear eficazmente como una herramienta de desarrollo para programadores. El ratón se enchufa en la puerta para palanca de mando del Commodore 64 y los iconos en pantalla se seleccionan mediante los tres botones existentes en el ratón



Crispin Thomas

Programas de utilidades

El paquete Magic Mouse no es un sistema operativo como el del Apple Macintosh, sino más bien una facilidad para el diseño de gráficos. Los cuatro programas que se entregan con el ratón son *Hi-res designer* (diseñador en alta resolución), *Sprite designer* (diseñador de sprites), *Icon designer* (diseñador de iconos) y *Mouse controller* (controlador del ratón).

Para poder ejecutar correctamente el software, primero se debe cargar (LOAD) el archivo del sistema y calibrar el cursor del ratón. El sistema basado en disco se carga automáticamente con un menú principal que permite seleccionar cualquiera de los cuatro programas de utilidades. El sistema basado en cassette es ligeramente distinto. Aquí el archivo del sistema forma parte del *Hi-res designer* y, por consiguiente, se ha de cargar este programa antes de poder utilizar cualquiera de los otros tres. Asimismo, el cursor debe posicionarse en la esquina inferior derecha de la pantalla antes de comenzar.

Hi-res designer es un programa de «caja de colores» de aspecto muy similar al software Koala Pad. Al cargarlo, la pantalla pasa a una serie de casilleros, cada uno de los cuales contiene una opción diferente, como Recuadro, Dibujar o Rellenar. Desplazando el cursor del ratón hasta uno de estos casilleros y pulsando el botón rojo del ratón, se selecciona dicha opción. De modo similar, para elegir ya sea el color de primer plano o el de fondo, uno simplemente debe llevar el cursor hasta las líneas de los 16 colores disponibles y seleccionar una. Para mover el «lienzo», se pulsa el botón azul.

Los paquetes *Icon* y *Sprite designer* son muy similares y en realidad sólo se diferencian en sus aplicaciones finales. Como la mayoría de los programas de esta naturaleza, el *Sprite designer* presenta una cuadrícula de 24 columnas por 21 filas. Los pixels se encienden pulsando el botón rojo cuando el cursor se halla sobre la posición adecuada. El sprite propiamente dicho aparece en una ventana situada en la esquina superior derecha de la pantalla, y cuando el cursor se sale de la cuadrícula en la esquina inferior derecha aparecen varias opciones. Éstas son para aplicaciones tales como ampliar el sprite, cambiar su color o pasar a un sprite diferente.

El *Icon designer* aparece casi idéntico, con una

cuadrícula en la esquina y el icono, que aparece arriba en una ventana. La diferencia existente entre ambos es que *Icon designer* se utiliza para construir gráficos definidos por el usuario, como, por ejemplo, paisajes, que se pueden guardar y volver a utilizar.

El último de los programas que suministra el software Magic Mouse es el *Mouse controller*. Este programa *no* hace nada por sí mismo, sino más bien proporciona el software activador que permite que sus programas se ejecuten desde el sistema. Aunque a primera vista éste es el menos útil de los cuatro programas, en realidad es el que posee el mayor potencial, puesto que estimulará al usuario a escribir su propio software para el ratón.

Características del cursor

La gran ventaja que tienen los sistemas basados en ratón sobre los digitalizadores es la uniformidad del cursor, que permite el posicionamiento exacto de las líneas y colores que se desea dibujar. Un digitalizador se basa en una cuadrícula de nudos interconectados; sin embargo, su resolución es inferior a la de la pantalla. Por tanto, cuando el «lápiz» se halla entre dos nudos, el ordenador no puede decidir cuál es la posición que se pretende y tiende, entonces, a saltar entre una y otra. Por supuesto, con frecuencia esto suele tener resultados desastrosos para la imagen que uno intenta construir. Cuando se utiliza el ratón, sin embargo, su cursor reacciona sólo a los cambios que se producen en sus potenciómetros, provocados por los movimientos de la bola, de modo que si no se mueve el ratón el cursor permanecerá quieto.

No obstante, el sistema no es perfecto. Algunos aspectos del *Hi-res designer* no son tan amables como los de otros paquetes. Por ejemplo, la instrucción RUB sólo permite borrar errores pixel a pixel. Ello lleva su tiempo y, si se borran pixels equivocados, se corre el riesgo de generar nuevos errores.

Un procedimiento mucho más acertado consiste en borrar todos los añadidos al lienzo desde la última vez que uno pasó por la pantalla de menú. Por supuesto, esto significa que en el proceso se borrará una buena cantidad de trabajo, pero al menos el usuario tendrá la posibilidad de empezar con una hoja en blanco.

El manual del Magic Mouse es extraordinariamente didáctico. Cada uno de los programas posee tanto una explicación completa sobre la forma en que trabaja el software de desarrollo, como información de programación para que usted incorpore el producto acabado a su propio software, incluyendo programas de muestra. Ello revela un buen trabajo de previsión por parte de SMC, a la que hay que felicitar por su exhaustivo y útil manual.

Si bien el Magic Mouse de SMC Supplies dista mucho de ser el Macintosh Emulator, la empresa obviamente ha desarrollado el software inicial como una herramienta para el programador, similar al ratón AMX o al software DR GEM. Sin duda alguna, la empresa espera que los programadores produzcan software adicional que consiguientemente aumente el atractivo del sistema. Puede que este enfoque no produzca ventas rápidas, pero, en cambio, es probable que asegure una larga vida al Magic Mouse.

John Clementson

Desarrollo artístico
Estas imágenes se crearon mediante el programa de aplicaciones *Hi-res designer* para usar con el SMC Magic Mouse. El programa admite las instrucciones usuales de que disponen esta clase de paquetes, incluyendo LINE, DRAW, FILL y CIRCLE. Se pueden utilizar hasta 16 colores diferentes para componer la imagen, y hay una amplia gama de distintos «pinceles», que producen líneas de diverso espesor y textura

MAGIC MOUSE

INTERFACE
Se enchufa en la puerta para palanca de mando del Commodore 64

SOFTWARE
Se entrega con cuatro programas de aplicaciones tanto en disco como en cassette

DOCUMENTACION
El manual ofrece detalles completos sobre cómo instalar el ratón, cómo usar el software existente y cómo incorporar los resultados en los propios programas del usuario

VENTAJAS
Parece bien construido y fiable, y los usuarios tendrán pocos problemas para transferir a sus propios programas los sprites y gráficos definidos por el usuario

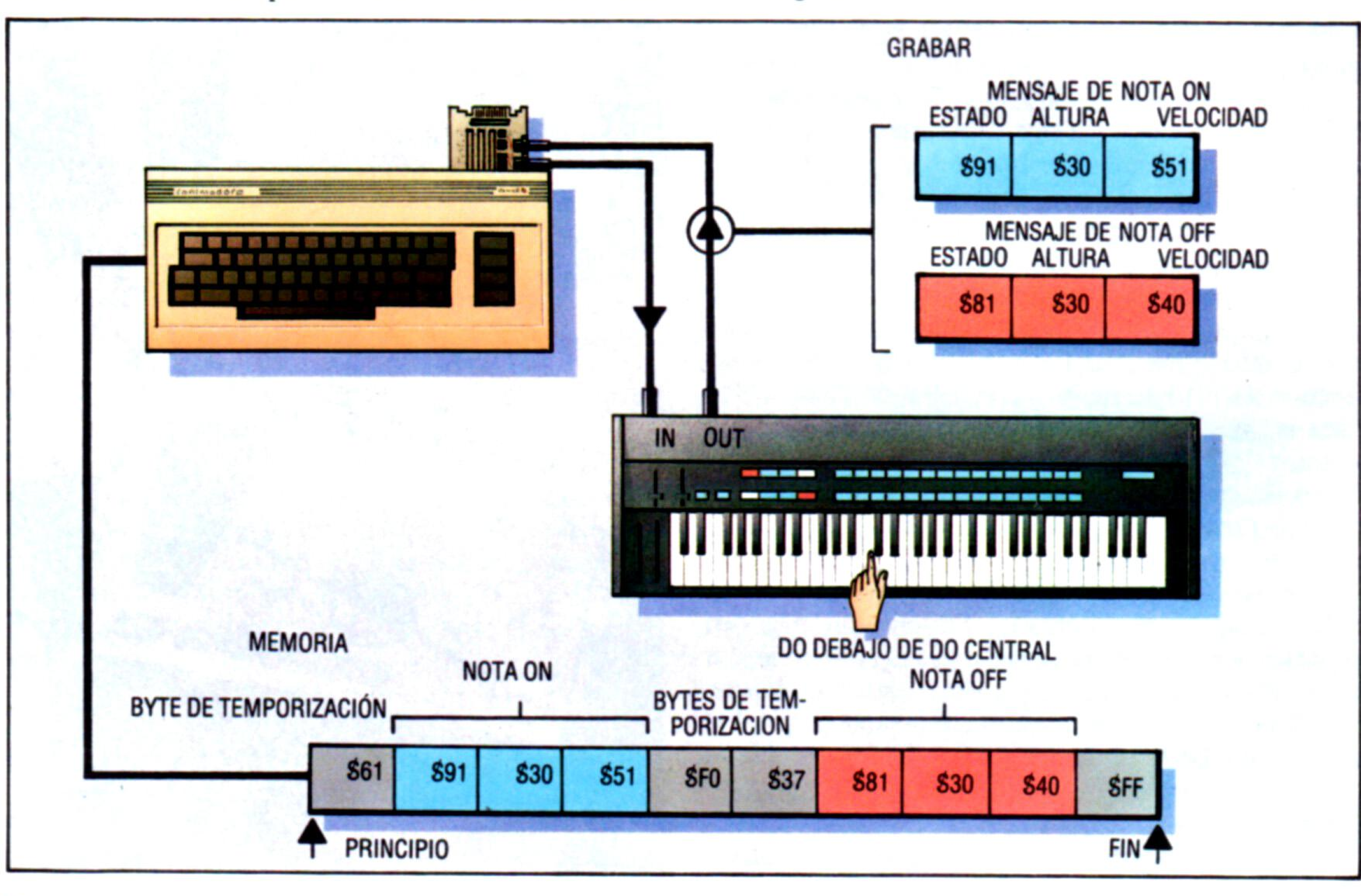
DESVENTAJAS
El paquete en conjunto carece del alcance de muchos otros programas para ratón, y se encontrará muy retrasado respecto a los paquetes más sofisticados

Alta fidelidad

Escribiremos un programa que permitirá que el ordenador actúe como un dispositivo digital grabador y reproductor en tiempo real

Cuando se pulsan las teclas de un teclado MIDI, o cuando se activa un dispositivo controlador como un mando de control de altura, estos «eventos» se transmiten en forma de bytes de datos al conector MIDI OUT de la parte trasera del sintetizador. nador una melodía tocada en tiempo real. Luego, en respuesta a una instrucción, el programa reproducirá la melodía desde la MIDI nuevamente hacia el sintetizador. Esto, por supuesto, es una grabación digital.

La voz de su micro
Nuestro diagrama ilustra cómo se grabaría una sola nota utilizando el programa *Grabación digital*. Durante la fase de grabación, el sintetizador envía un mensaje de *nota on* cuando se pulsa la tecla, y un mensaje de *nota off* cuando la misma se libera. Estos mensajes se almacenan en la memoria, junto con información de temporización que registra cuántos períodos de 2 milisegundos transcurren entre un evento y el siguiente. La información de temporización entre los mensajes de *nota on* y *nota off* en el ejemplo se retiene en dos bytes, mostrando cómo se maneja el desbordamiento del temporizador. Durante la reproducción, el programa utiliza la información de temporización para demorar la transmisión de los mensajes de nota, de modo que se reproduzca la misma «articulación»

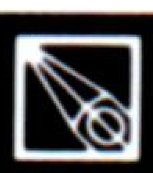


Gran parte de nuestro capítulo anterior lo dedicamos a ofrecer detalles acerca del significado de estos valores de bytes, para que usted pudiera escribir programas que aceptaran datos MIDI y los procesara. El ordenador también se puede utilizar como una fuente de datos MIDI, que se pueden transmitir a un sintetizador con el objeto de activar sus circuitos de generación de sonido. Un ejemplo obvio de este segundo uso es el de organizar el ordenador como un secuenciador, permitiendo entrar y editar en *tiempo de pasos* (programación de intervalos individuales dentro de una barra) frases musicales desde el teclado del ordenador. Ello se hace antes de la transmisión a través de la MIDI para activar el sintetizador que tocará la melodía. De hecho, es una labor relativamente simple: la MIDI es capaz de mucho más. ¡Piense en un secuenciador que utilice la totalidad de los 16 canales de la MIDI para controlar una matriz de diferentes teclados y una caja de ritmos!

Podemos combinar varios aspectos de la forma en que se puede utilizar un ordenador para que interactúe con instrumentos MIDI, asumiendo la tarea relativamente directa de diseñar un trozo de software que pueda grabar en la memoria del orde-

A pesar de que superficialmente es similar a la grabación analógica utilizando una grabadora de cintas y una cinta magnética, la grabación digital a través de la MIDI es radicalmente diferente. Cuando se efectúa una grabación en un trozo de cinta, los verdaderos sonidos musicales se codifican en patrones magnéticos en la superficie de la cinta, y la posición de estos patrones en la cinta determina el orden por el cual se han de reproducir los sonidos durante la reproducción, así como los intervalos de tiempo entre un sonido y el siguiente. Imaginemos que se están grabando dos notas de esta forma, suponiendo que entre éstas había un vacío de dos segundos. En este intervalo la cinta continuaría pasando las cabezas de cinta como si no se hubiera grabado nada. Cuando se reprodujera la cinta, la longitud de esta sección de cinta en blanco determinaría el intervalo de tiempo entre que el oyente escuchara la primera y la segunda nota.

En un sistema de grabación digital que utilice MIDI, las dos notas corresponderían a dos eventos de pulsaciones de teclas, cada uno de ellos generando un mensaje MIDI. El problema de una grabación en tiempo real es que debemos tener alguna forma de grabar el intervalo de tiempo entre que el



ordenador recibe el primero y el segundo mensaje MIDI. Utilizando la analogía de la grabación en cinta, la forma más obvia de hacerlo sería almacenando los mensajes en una matriz de la memoria, almacenando los valores de bytes «en blanco» para cada unidad de tiempo en el intervalo. Por tanto, durante la reproducción, las dos notas se tocarían con el intervalo de tiempo correcto. No es difícil comprender que este método de grabar eventos MIDI en tiempo real supone potencialmente un gran derroche de preciosa memoria.

Una alternativa para esta analogía simplista de una grabadora de cinta analógica consiste en grabar la información de tiempo en forma de bytes en lugar de en forma de valores de bytes «en blanco». Por ejemplo, si hay 50 unidades de tiempo entre la primera y la segunda nota, en la matriz de la memoria se entraría un byte de temporización que contuviera el valor 50, en vez de 50 bytes «en blanco».

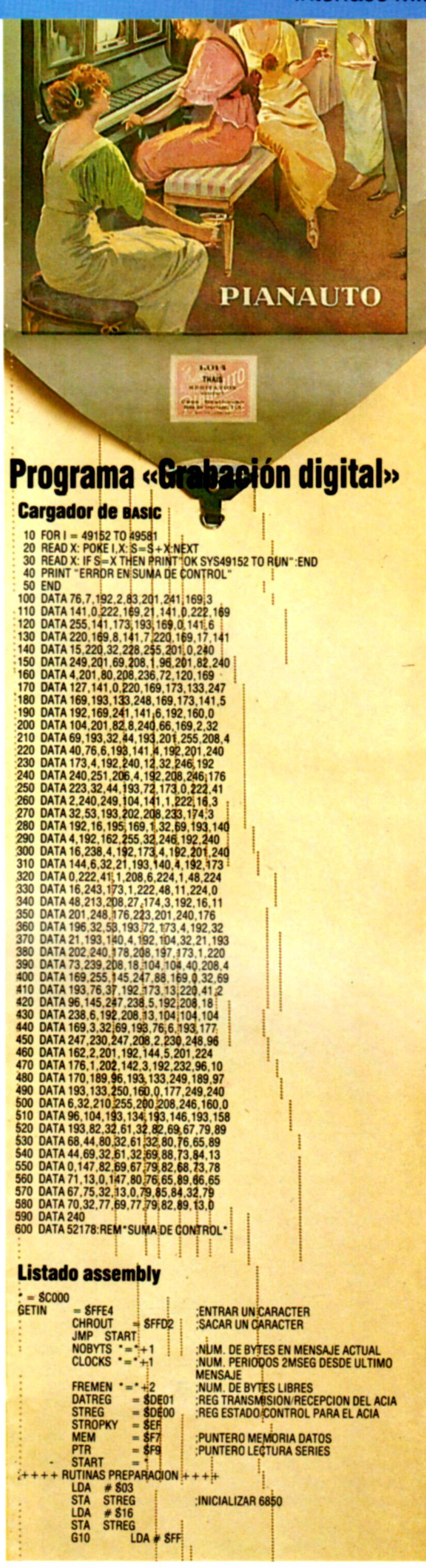
Ello supone un considerable ahorro de memoria, pero también dificulta la predicción de la longitud máxima de una melodía que se puede grabar en una cantidad de memoria dada. Las melodías que implican un rápido trabajo de los dedos, por ejemplo, requerirán más bytes para grabar la gran cantidad de mensajes de notas *on/off* transmitidos por el sintetizador que una pieza más lenta de la misma longitud. La utilización del mando de altura también tiende a devorar enormes cantidades de memoria, debido a la cantidad de mensajes MIDI que genera su operación.

Grabación digital

En el programa que ofrecemos necesitaremos, en consecuencia, alguna clase de contador para grabar los intervalos de tiempo entre los eventos. El contador que utilizamos es una variable llamada CLOCKS (relojes). La velocidad a la cual se incrementa este contador establece la precisión con la que se puede determinar la temporización y, por consiguiente, la fidelidad de la grabación. Al seleccionar el intervalo de tiempo óptimo hemos debido hacer un compromiso, optando por un intervalo de alrededor de dos milisegundos. Éste no es tan pequeño como para ser menor que los tiempos de ejecución de las rutinas, pero tampoco es tan grande como para incidir negativamente en la precisión de la grabación. Este intervalo de tiempo se denomina la *resolución* de la grabadora, dado que es el menor intervalo de tiempo que se puede grabar. El intervalo lo genera un temporizador de uno de los chips CIA o VIA del micro.

El formato del mensaje MIDI se puede ampliar ligeramente para tener en cuenta la temporización. Esto se hace colocando antes de cada mensaje MIDI un único byte que representa la cantidad de intervalos de tiempo que han transcurrido desde el último mensaje. Este único byte puede registrar hasta 239 unidades de tiempo y, por tanto, puede tener valores de entre $0 y $EF.

Si pasan 240 intervalos sin que se reciba un mensaje, se graba un mensaje de desbordamiento. Este mensaje es el byte $F0. Por último, se ha elegido un mensaje de un único byte, $FF, para señalar el final de la secuencia. Los valores de byte de $F1 a $FE están reservados para sus propios punteros de programa.



Programa «Grabación digital»

Cargador de BASIC

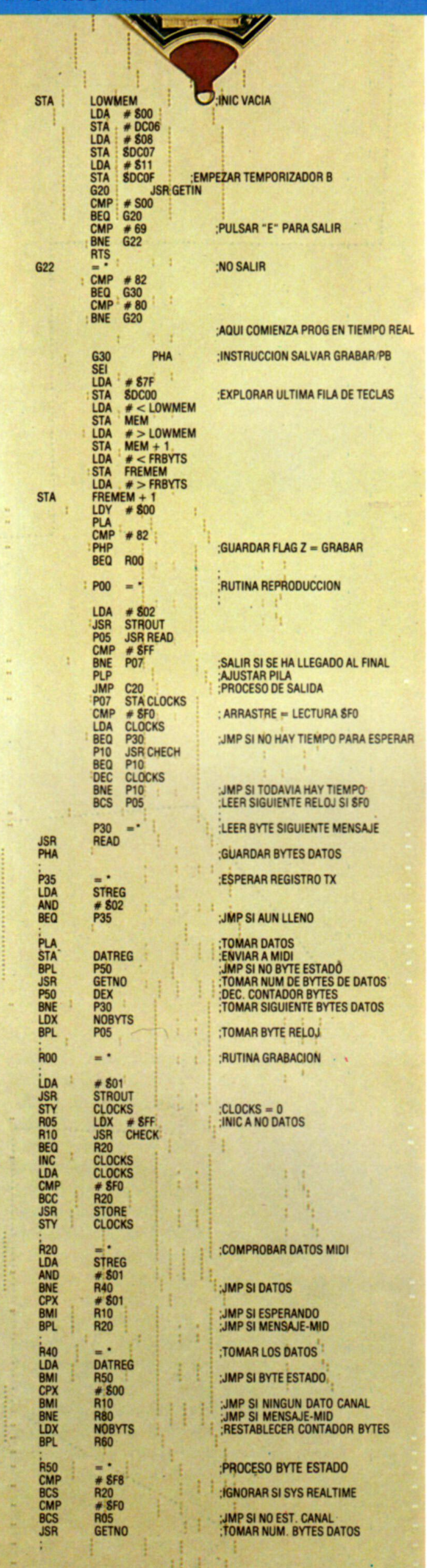
```
10 FOR I = 49152 TO 49581
20 READ X: POKE I,X: S=S+X:NEXT
30 READ X: IF S=X THEN PRINT"OK SYS49152 TO RUN":END
40 PRINT "ERROR EN SUMA DE CONTROL"
50 END
100 DATA 76,7,192,2,83,201,241,169,3
110 DATA 141,0,222,169,21,141,0,222,169
120 DATA 255,141,173,193,169,0,141,6
130 DATA 220,169,8,141,7,220,169,17,141
140 DATA 15,220,32,228,255,201,0,240
150 DATA 249,201,69,208,1,96,201,82,240
160 DATA 4,201,80,208,236,72,120,169
170 DATA 127,141,0,220,169,173,133,247
180 DATA 169,193,133,248,169,173,141,5
190 DATA 192,169,241,141,6,192,160,0
200 DATA 104,201,82,8,240,66,169,2,32
210 DATA 69,193,32,44,193,201,255,208,4
220 DATA 40,76,6,193,141,4,192,201,240
230 DATA 173,4,192,240,12,32,246,192
240 DATA 240,251,206,4,192,208,246,176
250 DATA 223,32,44,193,72,173,0,222,41
260 DATA 2,240,249,104,141,1,222,16,3
270 DATA 32,53,193,202,208,233,174,3
280 DATA 192,16,195,169,1,32,69,193,140
290 DATA 4,192,162,255,32,246,192,240
300 DATA 16,238,4,192,173,4,192,201,240
310 DATA 144,6,32,21,193,140,4,192,173
320 DATA 0,222,41,1,208,6,224,1,48,224
330 DATA 16,243,173,1,222,48,11,224,0
340 DATA 48,213,208,27,174,3,192,16,11
350 DATA 201,248,176,223,201,240,176
360 DATA 196,32,53,193,72,173,4,192,32
370 DATA 21,193,140,4,192,104,32,21,193
380 DATA 202,240,178,208,197,173,1,220
390 DATA 73,239,208,18,104,104,40,208,4
400 DATA 169,255,145,247,88,169,0,32,69
410 DATA 193,76,37,192,173,13,220,41,2
420 DATA 96,145,247,238,5,192,208,18
430 DATA 238,6,192,208,13,104,104,104
440 DATA 169,3,32,69,193,76,6,193,177
450 DATA 247,230,247,208,2,230,248,96
460 DATA 162,2,201,192,144,5,201,224
470 DATA 176,1,202,142,3,192,232,96,10
480 DATA 170,189,96,193,133,249,189,97
490 DATA 193,133,250,160,0,177,249,240
500 DATA 6,32,210,255,200,208,246,160,0
510 DATA 96,104,193,134,193,146,193,158
520 DATA 193,82,32,61,32,82,69,67,79,89
530 DATA 68,44,80,32,61,32,80,76,65,89
540 DATA 44,69,32,61,32,69,88,73,84,13
550 DATA 0,147,82,69,67,79,82,68,73,78
560 DATA 71,13,0,147,80,76,65,89,66,65
570 DATA 67,75,32,13,0,79,85,84,32,79
580 DATA 70,32,77,69,77,79,82,89,13,0
590 DATA 240
600 DATA 52178:REM"SUMA DE CONTROL"
```

Listado assembly

```
* = $C000
GETIN       = $FFE4                ;ENTRAR UN CARACTER
        CHROUT     = $FFD2         ;SACAR UN CARACTER
        JMP   START
        NOBYTS  *=*+1              ;NUM. DE BYTES EN MENSAJE ACTUAL
        CLOCKS  *=*+1              ;NUM. PERIODOS 2MSEG DESDE ULTIMO
                                   MENSAJE
        FREMEN  *=*+2              ;NUM. DE BYTES LIBRES
        DATREG     = $DE01         ;REG TRANSMISION/RECEPCION DEL ACIA
        STREG      = $DE00         ;REG ESTADO/CONTROL PARA EL ACIA
        STROPKY    = $EF
        MEM        = $F7           ;PUNTERO MEMORIA DATOS
        PTR        = $F9           ;PUNTERO LECTURA SERIES
        START      = *
;++++ RUTINAS PREPARACION ++++
        LDA   # $03
        STA   STREG                ;INICIALIZAR 6850
        LDA   # $16
        STA   STREG
        G10        LDA # $FF
```

Operación del programa
La operación del programa *Grabación digital* para el Commodore 64 es directa. La selección de la opción «grabar» grabará los eventos que lleguen a MIDI IN. La grabación se detiene ya sea cuando no se utiliza la memoria disponible o bien cuando se pulsa la barra espaciadora. La opción «reproducir» reproducirá luego la secuencia de eventos grabada en MIDI OUT. La reproducción continúa hasta que se pulsa la barra espaciadora, o bien al llegar al fin de la secuencia. Toda ulterior grabación se sobreescribirá en la anterior

Marcus Wilson-Smith



```
STA      LOWMEM              ;INIC VACIA
         LDA   # $00
         STA   # DC06
         LDA   # $08
         STA   $DC07
         LDA   # $11
         STA   $DC0F         ;EMPEZAR TEMPORIZADOR B
         G20        JSR GETIN
         CMP   # $00
         BEQ   G20
         CMP   # 69          ;PULSAR "E" PARA SALIR
         BNE   G22
         RTS
G22      = *                 ;NO SALIR
         CMP   # 82
         BEQ   G30
         CMP   # 80
         BNE   G20
                             ;AQUI COMIENZA PROG EN TIEMPO REAL

         G30        PHA      ;INSTRUCCION SALVAR GRABAR/PB
         SEI
         LDA   # $7F
         STA   $DC00         ;EXPLORAR ULTIMA FILA DE TECLAS
         LDA   # < LOWMEM
         STA   MEM
         LDA   # > LOWMEM
         STA   MEM + 1
         LDA   # < FRBYTS
         STA   FREMEM
         LDA   # > FRBYTS
STA      FREMEM + 1
         LDY   # $00
         PLA
         CMP   # 82
         PHP                 ;GUARDAR FLAG Z = GRABAR
         BEQ   R00

         P00   = *           ;RUTINA REPRODUCCION

         LDA   # $02
         JSR   STROUT
         P05   JSR READ
         CMP   # $FF
         BNE   P07           ;SALIR SI SE HA LLEGADO AL FINAL
         PLP                 ;AJUSTAR PILA
         JMP   C20           ;PROCESO DE SALIDA
         P07   STA CLOCKS
         CMP   # $F0         ; ARRASTRE = LECTURA $F0
         LDA   CLOCKS
         BEQ   P30           ;JMP SI NO HAY TIEMPO PARA ESPERAR
         P10   JSR CHECH
         BEQ   P10
         DEC   CLOCKS
         BNE   P10           ;JMP SI TODAVIA HAY TIEMPO
         BCS   P05           ;LEER SIGUIENTE RELOJ SI $F0

         P30   =*            ;LEER BYTE SIGUIENTE MENSAJE
JSR      READ
PHA                          ;GUARDAR BYTES DATOS
:
P35      = *                 ;ESPERAR REGISTRO TX
LDA      STREG
AND      # $02
BEQ      P35                 ;JMP SI AUN LLENO
:
PLA                          ;TOMAR DATOS
STA      DATREG              ;ENVIAR A MIDI
BPL      P50                 ;JMP SI NO BYTE ESTADO
JSR      GETNO               ;TOMAR NUM DE BYTES DE DATOS
P50      DEX                 ;DEC. CONTADOR BYTES
BNE      P30                 ;TOMAR SIGUIENTE BYTES DATOS
LDX      NOBYTS
BPL      P05                 ;TOMAR BYTE RELOJ
:
R00      = *                 ;RUTINA GRABACION
:
LDA      # $01
JSR      STROUT
STY      CLOCKS              ;CLOCKS = 0
R05      LDX   # $FF         ;INIC A NO DATOS
R10      JSR   CHECK
BEQ      R20
INC      CLOCKS
LDA      CLOCKS
CMP      # $F0
BCC      R20
JSR      STORE
STY      CLOCKS
:
R20      = *                 ;COMPROBAR DATOS MIDI
LDA      STREG
AND      # $01
BNE      R40                 ;JMP SI DATOS
CPX      # $01
BMI      R10                 ;JMP SI ESPERANDO
BPL      R20                 ;JMP SI MENSAJE-MID
:
R40      = *                 ;TOMAR LOS DATOS
LDA      DATREG
BMI      R50                 ;JMP SI BYTE ESTADO
CPX      # $00
BMI      R10                 ;JMP SI NINGUN DATO CANAL
BNE      R80                 ;JMP SI MENSAJE-MID
LDX      NOBYTS              ;RESTABLECER CONTADOR BYTES
BPL      R60
:
R50      = *                 ;PROCESO BYTE ESTADO
CMP      # $F8
BCS      R20                 ;IGNORAR SI SYS REALTIME
CMP      # $F0
BCS      R05                 ;JMP SI NO EST. CANAL
JSR      GETNO               ;TOMAR NUM. BYTES DATOS
:
```

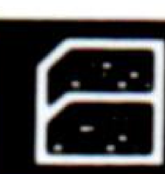
```
R60      = *                 ;GRABAR LOS DATOS
PHA
LDA      CLOCKS
JSR      STORE
STY      CLOCKS              ;RESTABLECER CONTADOR
PLA
:
R80      = *                 ;GRABAR DATOS MIDI
JSR      STORE
DEX
BEQ      R10
BNE      R20
:
CHECK    = *
:
LDA      $DC01
EOR      # STOPKY            ;NO AFECTA AL ARRASTRE
BNE      C40
PLA
PLA                          ;AJUSTAR PILA
PLP                          ;COMPROBAR REC/PB
BNE      C20                 ;JMP SI PLAYBACK
LDA      # $FF
STA      (MEM),Y
C20      = *
CLI
LDA      # $00
JSR      STROUT
JMP      G20                 ;ESPERAR SIGUIENTE INSTRUCCION
:
C40      = *                 ;COMPROBAR TEMPORIZADOR B
LDA      $DC0D
AND      # $02               ;FLAG Z = TIEMPO NO AGOTADO
RTS
:
STORE    = *                 ;ALMACENAR DATOS EN MEMORIA
:
STA      (MEM),Y
INC      FREMEM              ;COMPROBAR MEMORIA LIBRE
BNE      POINT
INC      FREMEM + 1
BNE      POINT
PLA
PLA
PLA
LDA      # $03               ;MENSAJE DE ERROR
JSR      STROUT
JMP      C20                 ;RESTAURAR PROGRAMA
:
READ     = *                 ;LEER DATOS DE LA MEMORIA
:
LDA      (MEM),Y
:
POINT    = *                 ;ACTUALIZAR PUNTERO MEMORIA
:
INC      MEM
BNE      P20
INC      MEM + 1
P20      RTS
:
GETNO    = *                 ;TOMAR NUM. BYTES DE DATOS
:             PARA ESTADO EN ACC.
:
LDX      # $02               ;INIC A 2 BYTES DATOS
CMP      # $C0
BCC      N10
CMP      # $E0
BCS      N10
DEX                          ;1 BYTE DATOS SI C0<=A<E0
N10      STX   NOBYTS
INX                          ;SUMAR 1 PARA ESTADO
RTS
:
STROUT   = *                 ;IMPRIMIR SERIE
:
ASL      A
TAX
LDA      MESTAB,X
STA      PTR
LDA      MESTAB+1,X
STA      PTR+1
LDY      # $00
M10      LDA  (PTR),Y
BEQ      M70
JSR      CHROUT
INY
BNE      M10
M70      LDY  #$00
RTS
:
:
MESTAB   = *                 ;TABLA MENSAJES
:
.WORD    MESS0
.WORD    MESS1
.WORD    MESS2
.WORD    MESS3
:
MESS0    .BYTE 'R = GRABAR, P = TOCAR, E = SALIR',$0D,0


MESS1    .BYTE 147,'GRABANDO',$0D,0


MESS2    .BYTE 147,'PLAYBACK',$0D,0


MESS3    .BYTE 'FUERA DE MEMORIA',$0D,0


LOWMEM   = *                 ;COMIENZO DE MEMORIA DATOS
FRBYTS   = LOWMEM - $D000
```



Lista activa

Concluiremos nuestra serie dedicada al LISP estudiando otras funciones de las que suele disponer el lenguaje

Para disponer de una notación uniforme que pueda tratar fácilmente con identificadores, caracteres, números y listas de la misma manera, el LISP se refiere a todo mediante *punteros*. Éstos se pueden simular fácilmente en cualquier lenguaje, incluyendo al BASIC, por ejemplo, si usted imparte la instrucción:

```
POKE 100,55 : POKE 99,100 (la mayoria de los
  micros)
```

o:

```
?100 = 55 : ?99 = 100 (BBC Micro/Electron)
```

las posiciones 100 y 99 contendrán los valores dados. Se puede considerar que el valor de la posición 99 es un puntero a la posición 100. De modo, pues, que si diéramos la instrucción:

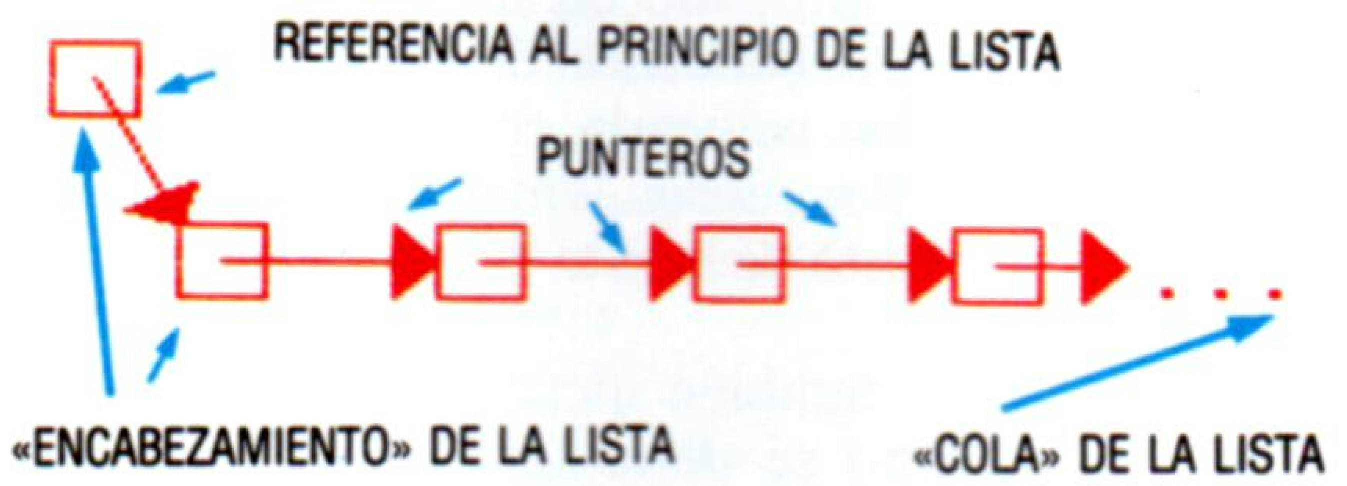
```
PRINT PEEK (PEEK(99))
```

o:

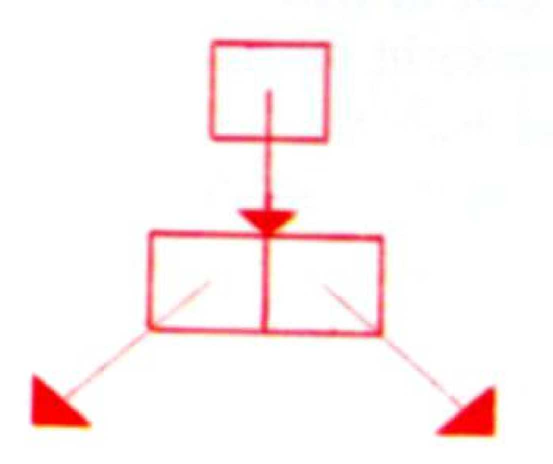
```
PRINT ?(?99)
```

obtendríamos 55 como respuesta. Observe que en algunos micros las posiciones 99 y 100 podrían estar en ROM, de modo que no cambiarían.

Por supuesto, es posible crear punteros hacia cualquier lugar de la memoria, pero si la posición se halla por encima de 255 será necesario combinar dos bytes para obtener la escala entre 0 y 65535. Ahora, si suponemos que, a su vez, el valor 55 es un puntero que señala a otra posición, y así sucesivamente, obtendríamos la estructura de lista indicada.

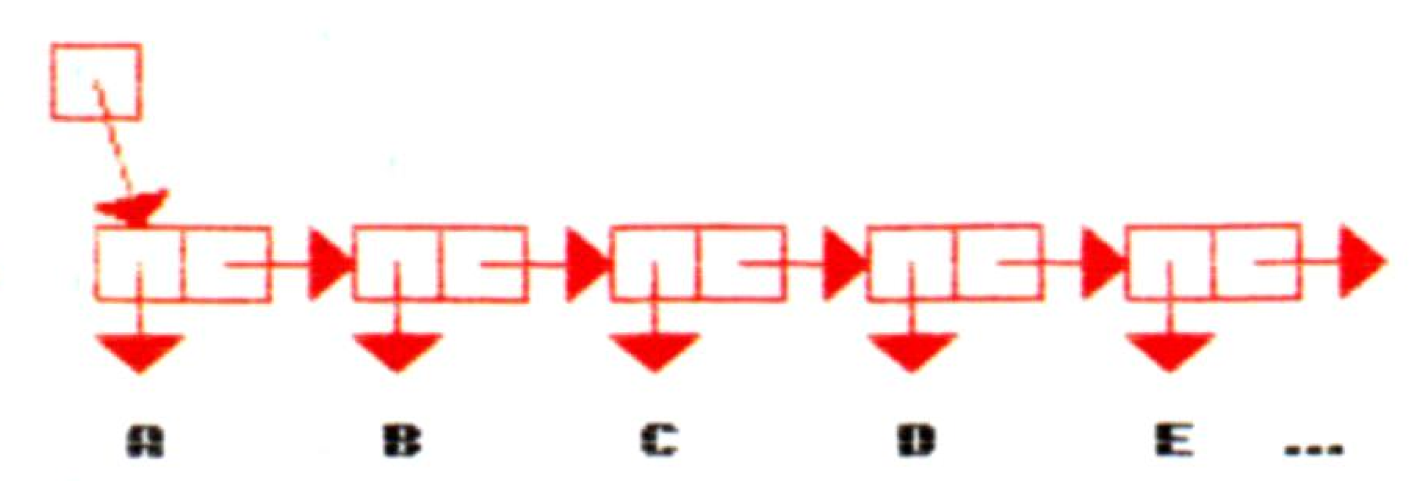


Lamentablemente, ésta aún sirve de muy poco, puesto que todavía no retiene ninguna información. Ello se consigue combinando posiciones para otorgarle dos punteros a cada nudo.



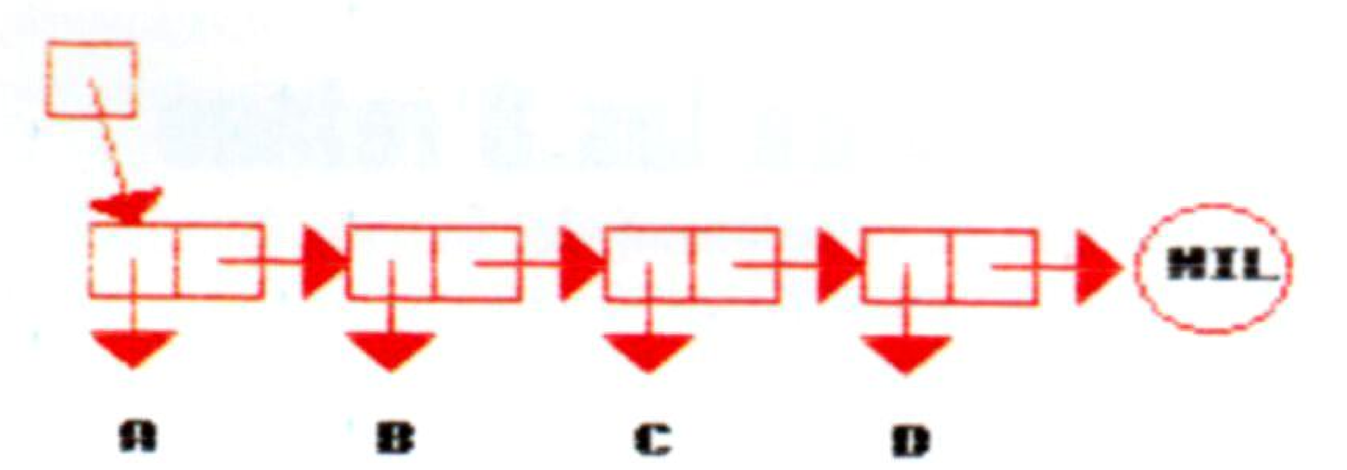
Ahora utilicemos esto para representar la lista:

```
(A B C D E...)
```



Esto está muy bien para listas infinitas, pero debemos ser capaces de detenerlas. Ello se consigue fácilmente haciendo que el puntero final aluda a la lista vacía (o construcción NIL), que ya hemos visto antes. Por tanto, obtenemos la lista finita:

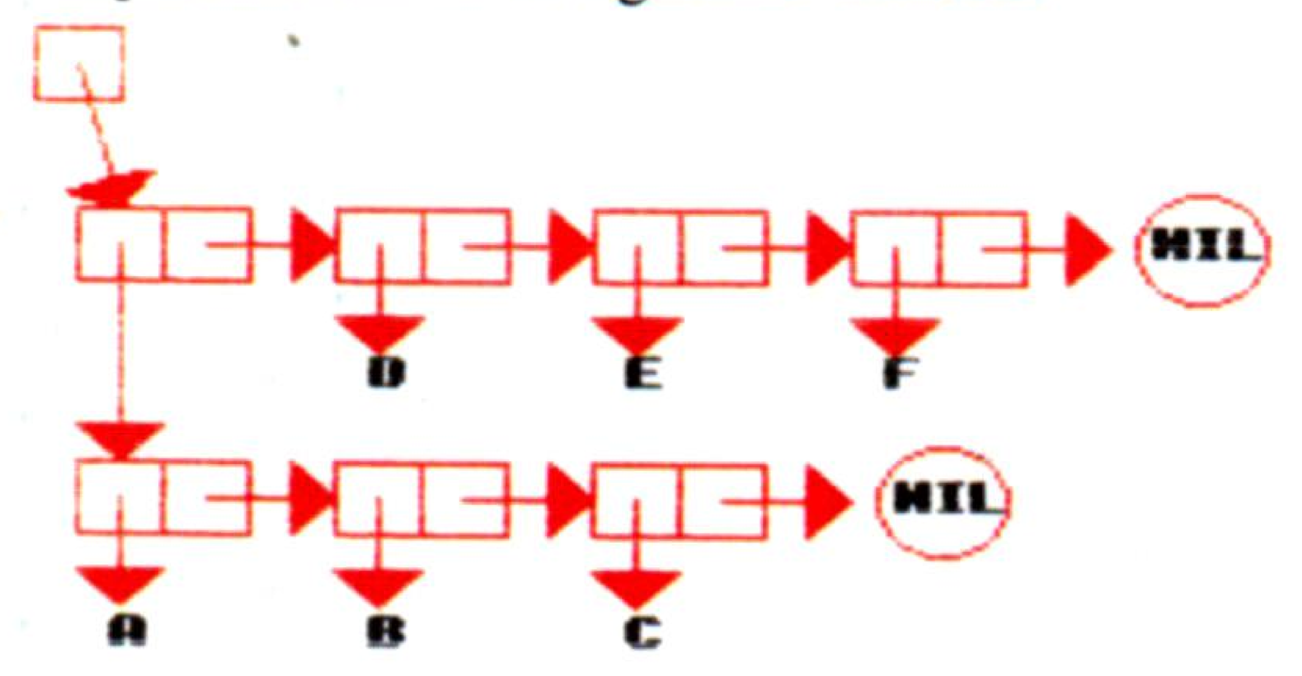
```
(A B C D)
```



Una lista que contiene una lista significa, simplemente, que uno de los punteros de la izquierda señala hacia una nueva lista, en lugar de a un átomo, como en el caso anterior. De modo que la lista:

```
((A B C)D E F)
```

se representaría de la siguiente manera:



A estas alturas ya habrá de estar clara la forma en que trabajan las funciones CAR y CDR. CAR (el encabezamiento de su argumento) no es más que el puntero de la izquierda del primer nudo de su argumento. Por el contrario, CDR (todos a excepción del encabezamiento de su argumento) es simplemente el puntero de la derecha del primer nudo. Así, si llamáramos a la lista completa L, obtendríamos:

```
CONS((CAR L)(CDR L))
```

que da como resultado la lista original L.

Podemos, entonces, ver que CONS es un método para construir listas. Convierte a sus dos argumentos en una nueva lista y devuelve como resultado un puntero al primer elemento de la nueva lista. Mediante CONS se puede crear cualquier estructura

de lista, si bien la cantidad de llamadas CONS puede resultar tediosa, de allí la función abreviada LIST.

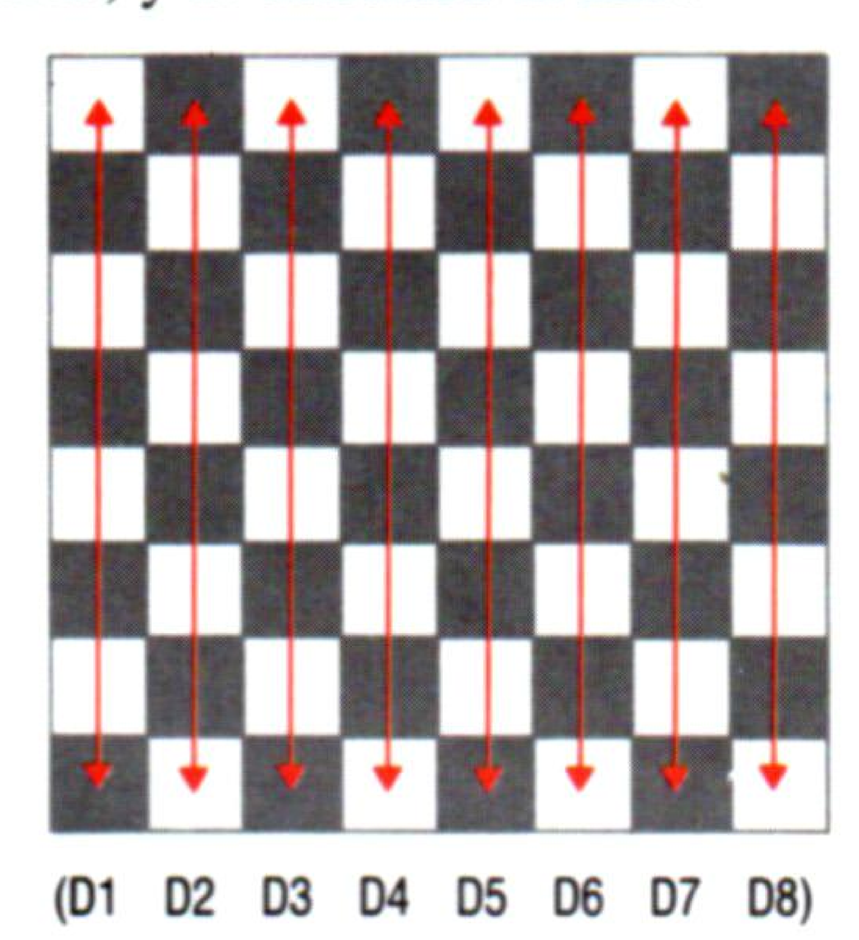
Hemos de destacar que siempre es preferible añadir nueva información a la cabeza de la lista utilizando CONS. Al añadirla a la cola de una lista, es necesario crear una copia de la lista original, pero con el puntero final original señalando hacia el nuevo elemento.

Algunas versiones de LISP poseen una instrucción para esto de forma automática (APPEND), pero ésta no es tan eficaz como CONS. Si la velocidad es esencial, entonces algunas veces es posible utilizar una instrucción para cambiar el puntero requerido en la lista original, en lugar de generar una copia; pero con frecuencia los puristas condenan el empleo de esta clase de «disparates».

Para concluir esta serie, veamos un ejemplo completamente trabajado de un problema muy conocido. El problema consiste en colocar ocho reinas en un tablero de ajedrez normal, de modo tal que ninguna de las reinas se amenace entre sí. Obviamente, ha de haber exactamente una reina en cada columna y una en cada fila. El problema consiste en ordenar tanto las filas como las columnas de modo que ninguna de las reinas se intersecte a lo largo de alguna diagonal.

Problema de las 8 reinas

Imagine que cada columna del tablero de ajedrez contiene una reina. Éstas se ordenarán (del 1 al 8) según sus lugares en una lista llamada REINAS. La fila de la reina (especificada por el valor de REINAS) será variable, y lo calculará el LISP.



REINAS (D1 D2 D3 D4 D5 D6 D7 D8)

El programa comienza con la lista REINAS vacía. En cada etapa se colocará una nueva reina en la fila 1 al comienzo de la lista. La posición de esta reina, y posiblemente de otras, se corregirá (función ALTER) hasta que todas las reinas se encuentren en posiciones seguras. Esta nueva lista se convertirá en REINAS, y se añadirán nuevas reinas al comienzo, y así sucesivamente. Esto continúa hasta que la lista contenga ocho reinas, en cuyo momento se las imprimirá mediante IMP_TABLERO. La función inicial, a la que llamaremos RESOLVER, se puede escribir así:

```
(DEFUN RESOLVER ()
  (SETQ REINAS ())
  (LOOP(UNTIL EQ (LONGITUD (REINAS)8))
    (SETQ REINAS (ALTER (CONS 1 REINAS))))
  (IMP_TABLERO' (DT DC DA D R RA RC
  RT)REINAS))
```

Aquí se puede ver que RESOLVER, sin ningún argumento, establece originalmente la lista vacía REINAS. Luego la función LOOP establece REINAS en el nuevo valor de una lista ALTERada de una reina en la fila 1, seguida por la lista anterior. Este bucle (LOOP) se ejecuta hasta (UNTIL) que la LONGITUD de REINAS sea equivalente a 8. Entonces se imprimirá la lista final.

Aquí hemos utilizado tres nuevas funciones. LONGITUD simplemente devuelve el número de elementos de su argumento, y se define así:

```
(DEFUN LONGITUD (L (N))
  (SETQ N 0)
  (LOOP (WHILE L N)
    (SETQ N (ADD1 N))
    (SETQ L (CDR L)) ))
```

El argumento N, dado entre paréntesis, es la forma que tiene el LISP Acornsoft de mostrar argumentos opcionales. En este caso, N no se le pasa nunca a la función y, por tanto, actúa simplemente como una variable local. La otra función, IMP_TABLERO, combina los elementos de sus dos listas de argumentos para producir una salida comprensible. La función es:

```
(DEFUN IMP_TABLERO (FILAS COLS)
  (COND ((NULL FILAS)T)
   (T (PRINTC (CAR FILAS)(CAR COLS))
    (IMP_TABLERO (CDR FILAS)(CDR   COLS)) )))
```

Se da por sentado que ambas listas son del mismo tamaño, de modo que la función termina si la primera condición es verdadera (una lista vacía). De lo contrario, se imprime el primer elemento de cada lista (con un retorno del carro) y la función se llamará a sí misma para el resto de cada lista.

La tercera función, ALTER, es la que hace todo el trabajo. Se define del siguiente modo:

```
(DEFUN ALTER (REINAS)
  (COND ((EQ (CAR REINAS)9)
  (ALTER (CONS (ADD1  (CADR REINAS))
                      (CDDR REINAS) )))
  ((SEGURAS REINAS)   REINAS)
  (T (ALTER (CONS (ADD1 (CAR REINAS))
                        (CDR REINAS) )))))
```

ALTER funciona comprobando en primer lugar si la reina colocada más recientemente se ha salido por arriba del tablero (fila 9). De ser así, se habrán de mover una o más de las reinas colocadas previamente.

Recuerde que esta reina comenzaba en la fila 1 y que se habrá comprobado en todas las otras filas (mediante la tercera condición de ALTER), de modo que será imposible colocarla con el tablero en la situación actual. Por consiguiente, ALTER se llama a sí misma con el resto de la lista REINAS (todo menos la reina actual).

Al hacer esto, también incrementa la fila de la última reina. Esto se denomina *retroceso* (*backtracking*), y es bastante común en los algoritmos para ordenador de este tipo. Si la segunda condición resulta TRUE (verdadera), entonces todas las reinas situadas en el tablero actual estarán SEGURAS y se devolverá la lista. De lo contrario, se incrementará la posición de la reina colocada en última instancia y se volverá a hacer otro intento.

La única función aún sin definir es SEGURAS, que ha de comprobar que todas las reinas del tablero actual estén seguras. Dado que la posición en la lista REINAS define la columna y que no hay dos



reinas que tengan la misma posición en la lista, no existe necesidad alguna de comprobar columnas que se intersecten. Esto nos deja con la tarea de comprobar sólo las filas y las diagonales. Si suponemos que tenemos una función NOJAQUE que devuelve TRUE (verdadero) si ninguna reina tiene en «jaque» a ninguna de las reinas precedentes, entonces SEGURAS se puede definir así:

```
(DEFUN SEGURAS (REINAS)
  (COND ((NULL REINAS)T)
    (T (AND (NOJAQUE (CAR REINAS)
                     (CDR REINAS) 1)
      (SEGURAS (CDR   REINAS)) ))))
```

Tal vez resulte más fácil comprenderlo definiendo la palabra SEGURAS en términos de sí misma: «La primera reina de la lista REINAS NO tiene en JAQUE a ninguna de las reinas del resto de la lista, AND (y) todas las reinas del resto de la lista están SEGURAS.»

Ahora sólo nos resta definir la función NOJAQUE, que habrá de devolver TRUE si a lo largo de las filas y de las diagonales no se produce ninguna intersección. Ésta será así:

```
(DEFUN NOJAQUE (NUEVA RESTO COL)
  (COND ((NULL RESTO) T)
    (T (AND (NOT (EQ NUEVA (CAR RESTO)))
      (NOT (EQ COL
        (ABS (DIFFERENCE NUEVA (CAR RESTO)))))
      (NOJAQUE NUEVA (CDR RESTO)(ADD1 COL)) ))))
```

Aquí, NUEVA es la reina colocada en último lugar, y RESTO es la lista de las otras reinas del tablero. Suponiendo que en éste haya otras reinas (NULL RESTO es FALSE), la función AND devuelve verdadero si:

1. La fila de la reina actual no es la misma que la fila de la primera reina del resto de la lista.
2. Las reinas no se hallan en la misma diagonal. Esto se comprueba asegurando que la diferencia de fila ABSoluta no sea la misma que la diferencia de COLumna, donde COL retiene la cantidad de columnas que separan a la reina actual de la reina siguiente de la lista (inicialmente 1). En el capítulo anterior ya habíamos definido a ABS como:

```
(DEFUN ABS (NUMERO)
  (COND ((MINUSP NUMERO)(MINUS NUMERO))
    (T NUMERO) ))
```

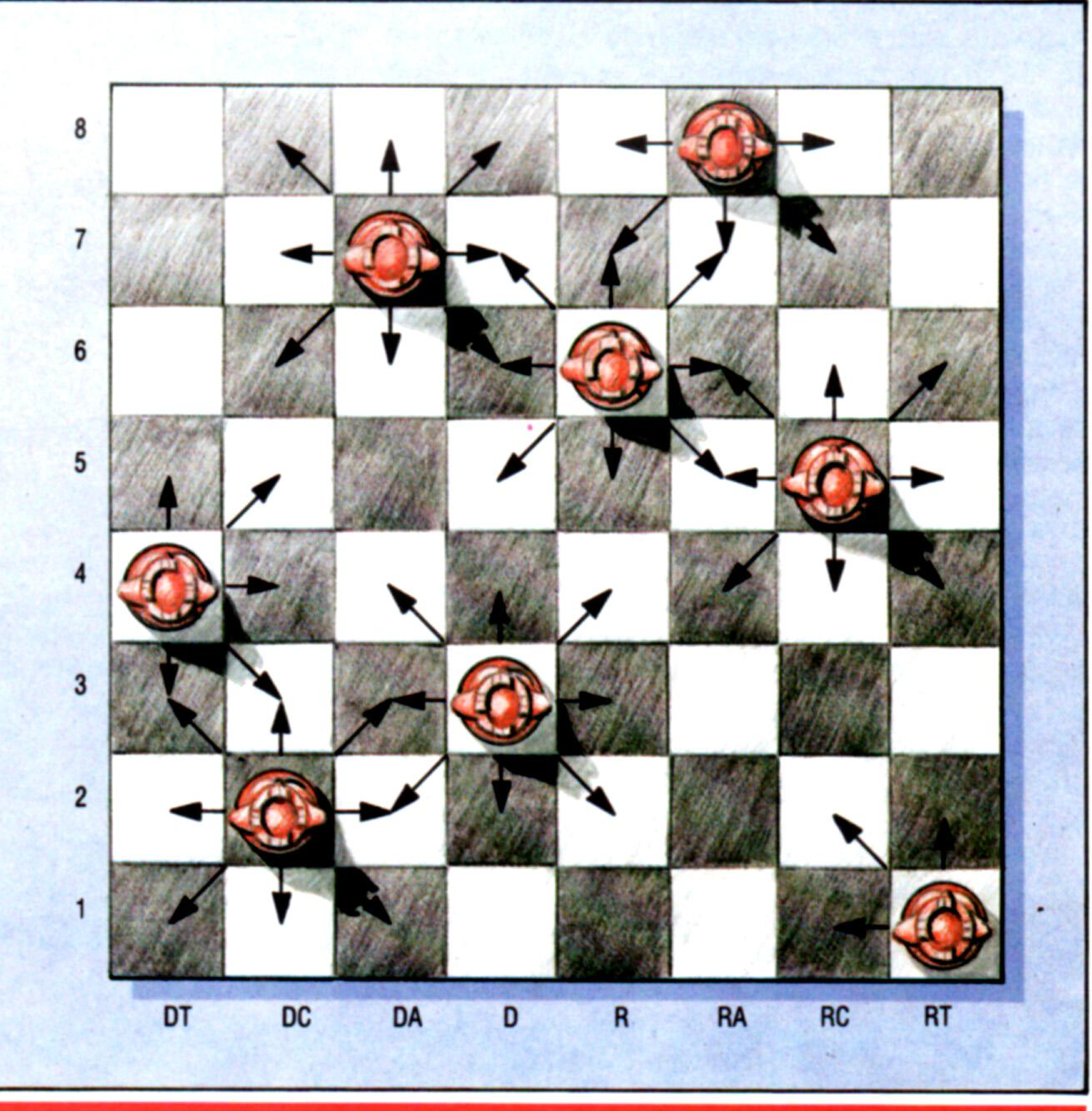
3. 1 y 2 son verdaderos para la reina actual y la siguiente reina de la lista. Esto lo comprueba la función llamándose a sí misma con todos los elementos de RESTO menos el primero e incrementando la diferencia de COLumna.

Ahora ya hemos completado las definiciones necesarias y en la ilustración vemos cómo se ejecutan.

Si usted posee una implementación de LISP, quizá le agrade probar estas rutinas. Recuerde que se han escrito utilizando LISP Acornsoft, si bien no será necesario modificarlas demasiado para que se ejecuten en otros sistemas. El retroceso se puede volver complicado, pero usted puede seguirlo fácilmente imprimiendo la lista REINAS dentro de la función RESOLVER, justo antes de cada llamada a ALTER.

El enfoque de programación de arriba abajo para el «problema de las 8 reinas» ilustra lo fácil que es definir programas en LISP sobre una base de función por función. Por ejemplo, hemos utilizado SEGURAS en ALTER antes de escribirla. Este ejemplo final también debe servir para demostrar la gran versatilidad que posee el LISP.

Un problema real
El problema de las 8 reinas es un ejemplo perfecto de un puzzle que se puede resolver rápidamente merced a las facilidades del LISP para manipulación de listas y funciones recursivas. Antes de examinar atentamente la solución que le ofrecemos en estas páginas, tal vez quiera colocar 8 reinas en un tablero de ajedrez de modo tal que cada reina esté a salvo de ser capturada por alguna de las otras. No se requiere conocimiento alguno de ajedrez, a excepción de que una reina (D) puede moverse en línea recta (diagonal, vertical u horizontalmente) a través de cualquier número de cuadrados, capturando cualquier pieza que encuentre en su camino



Ojo al microdrive

Las rutinas que controlan el funcionamiento de los microdrives del Spectrum son de gran utilidad para el programador en lenguaje máquina

Los accesos al microdrive se llevan a cabo bien por medio de los códigos de enganche ya vistos, o bien llamando directamente a las rutinas ROM de la Interface 1. Pero esto último no es muy recomendable, toda vez que pueden existir varias ROM de la Interface 1. Examinaremos, pues, los trece códigos de enganche dispuestos para que podamos usar el microdrive.

Antes de hacer este examen vamos a revisar algunos principios del manejo de un microdrive. Una vez formateado un cartucho, queda dividido en 255 *sectores,* que pueden ser considerados como las unidades físicas de almacenamiento de información. Cada sector se compone de un encabezamiento (datos que, entre otras cosas, incluyen el nombre del cartucho), un bloque de datos, que contiene el nombre del fichero que está usando ese sector, algunos otros datos y un registro de 512 bytes. El formateo pone marcas a todos los sectores de la cinta que no se pueden utilizar.

Al igual que todas las demás comunicaciones con los dispositivos de E/S del sistema del Spectrum, la comunicación microdrive-Spectrum se lleva a cabo por medio de un canal. Para los microdrives, éste toma la forma de un área de memoria de 295 bytes. Asociado a un canal de microdrive existe un mapa de 32 bytes para microdrives, establecido por medio del sistema operativo para indicar a las diferentes rutinas cuál de los sectores queda libre para su uso. Éste contiene una indicación sobre los sectores no utilizables y los que ya han sido utilizados para el almacenamiento de más información.

El canal se almacena en el área de memoria llamada *microdrive channel area* (área del canal de microdrive). Siempre que se establece un canal, la memoria que está entre el final del área del canal de microdrive y STKEND se desplaza hacia arriba dentro de la memoria para dejarle espacio. Igualmente, cuando ya no se necesita el canal, se cierra y baja de nuevo dicha memoria.

Los canales del microdrive son establecidos por el sistema de dos maneras: la *explícita,* cuando se requiere un fichero por la instrucción OPEN, y la *implícita*, cuando el sistema tiene que abrir un canal para realizar una operación. Es esto lo que hace la instrucción SAVE del microdrive: la apertura del canal se dice «implícita» dado que se sobreentiende que el canal debe estar abierto para ejecutar la instrucción, aun cuando no lo empleemos directamente. Un canal abierto explícitamente ha de ser cerrado manualmente, mientras que el cierre es automático en un canal abierto de forma implícita.

Ofrecemos en este capítulo un gráfico que muestra la estructura del canal del microdrive. Notará sin duda la semejanza entre los primeros bytes de este canal y los canales que establecemos para enviar información a la pantalla o leer datos desde el teclado. Buena parte de esta información no es estrictamente necesaria para el programador, ya que generalmente no hay que preocuparse de cómo se accede directamente a la información del canal del microdrive.

La dirección del mapa del microdrive está contenida en el canal, y podemos averiguar el espacio disponible en un cartucho por medio del examen del mapa de este cartucho. De hecho es así cómo la función del BASIC CAT proporciona información sobre la cantidad de espacio que queda en el cartucho. Si desea explorar el mapa del microdrive, pruebe la siguiente rutina (pulse previamente NEW).

```
 10 OPEN #5;"M";1;"testprog"
 20 start=PEEK(23870)+256*PEEK(23871)
 30 FOR I=start TO start+31
 40 LET sector=PEEK(I)
 50 FOR J=1 TO 8
 60 IF sector/2=INT(sector/2) THEN PRINT "0";
 70 IF sector/2<>INT(sector/2) THEN PRINT "1";
 80 LET sector=sector/2:LET sector=INT(sector)
 90 NEXT J
100 PRINT
110 NEXT I
120 CLOSE #5
```

Al ejecutar este programa, el 1 significa sector usado o no utilizable y el 0 sector libre. Puede que le interese alterar este programa para visualizar la cantidad de espacio libre en el cartucho.

Respecto a la sección enorme, de 512 bytes, para datos dentro del dibujo sobre la estructura de los canales, el OS lo llena con los datos a grabar. Los datos se escriben en el cartucho bien cuando el buffer está lleno, bien cuando se ejecuta una instrucción CLOSE# (o su código de enganche equivalente). En el último caso, el buffer es escrito en el cartucho quedando ese registro marcado como un EOF (*end of file:* fin de archivo). Una tercera posibilidad de escritura la da el empleo de un código de enganche, como veremos. Bajo condiciones normales, se escribirá en el cartucho a intervalos de 512 bytes.

Analicemos ahora los códigos de enganche empleados para controlar el microdrive.

Códigos de enganche del microdrive

- *Código de enganche 33:* enciende el motor en una unidad determinada del microdrive («selección del drive», suele decirse). Los microdrives están numerados de uno a ocho. La rutina puede servir

también para desconectar los motores de todos los microdrives. Y aquí es importante notar que la rutina puede ejecutar un Return con las interrupciones desactivadas. Así que, si usted emplea esta rutina, a su vuelta lo primero que habrá de hacer es generar la instrucción EI.

El empleo de la rutina es sencillo. Primero hay que disponer el registro A para que retenga el número del drive que desea poner en marcha. Después empleará RST#8 para llamar al código de enganche. Poniendo el valor 0 en el registro A se desconectarán los drives. De otra forma también pueden desconectarse: generando un error en BASIC. Los errores pueden generarse por medio de este código de enganche de dos maneras: tratando de poner en marcha un drive no conectado, o bien si se enciende un drive que no contenga un cartucho formateado.

- *Código de enganche 34:* Esta utilísima llamada nos permite abrir un fichero en un cartucho. Pronto estudiaremos cómo establecerla. Pero una vez llamada, la rutina examinará en el drive si existe un fichero con el nombre que usted le ha dado. Si encuentra uno, el fichero será abierto para *lectura.* Si no lo encuentra, será abierto para *escritura.* Esto se indica con el estado del flag de Escritura/Lectura del canal que establece esta llamada: el bit 0 valdrá 0 para leer un fichero y 1 para escribirlo.

Un fichero puede abrirse varias veces para lectura, y cada vez que se abra se establecerá un canal, incluso cuando todavía esté activo el canal previamente establecido.

Veamos lo que ocurre cuando se emplea esta llamada. Las variables de sistema que la Interface 1 establece son necesarias, de modo que si usted duda si están o no presentes habrá de emplear el código de enganche 49 para establecerlas. El número del drive se almacena entonces en las direcciones 23766 y 23767, con el byte *lo* primero. Se establecen entonces las direcciones 23770 y 23771 para contener la longitud del nombre que usted le ponga al fichero, y las direcciones 23772 y 23773 contienen la dirección del nombre del archivo en la memoria. En ambos casos los valores se almacenan con el byte *lo* primero. Un lugar útil para almacenar el nombre del fichero temporalmente es el buffer de la impresora.

Antes de emplearse esta rutina debe ejecutarse una instrucción EXX y guardarse en la pila el contenido del par de registros HL, y antes de volver al BASIC éstos deben ser restablecidos. Se llama entonces al código de enganche con RST#8 del modo acostumbrado. Al retorno, debe haber sido establecido un canal en el área de canal del microdrive y el registro IX debe contener la dirección de inicio de este canal. Los posibles errores que pueden cometerse en esta llamada son:

1. El archivo que usted desea abrir no es un archivo de datos.
2. El primer sector de un archivo que se abre para lectura no puede ser hallado.
3. Se ha tratado de abrir dos veces un archivo *write* (de sólo escritura).

- *Código de enganche 35:* Nos permite cerrar un fichero de microdrive que fue abierto con el código de enganche 34. Si el fichero se abrió para escritura, cualquier dato que se halle en el área de datos del canal todavía no enviado es escrito en el cartu-

0	Contiene &0008
2	Contiene &0008
4	"M"=explícito, "M"+128=implícito
5	Dirección rutina salida ROM Sombra
7	Dirección rutina entrada ROM Sombra
9	CHLENG Contiene 595 decimal
11	CHBYTE Puntero de pos. actual en el buffer
13	CHREC Núm. de reg. dentro arch.
14	CHNAME Nombre archivo (10 caract.)
23	Bit 0 encend.=escr., apag.= lect.
24	Bits del 1 al 7 activos y sin usar
25	CHDRIVE Núm. drive (del 1 al 8)
26	CHMAP Dirección del mapa del microdrive asociado al canal
28	Zona de trabajo, encabezamiento
40	HDFLAG Bit 0 encend.=encab.
41	Número sector (del 0 al 255)
	No utilizado (contiene &0E31)
44	Nombre cartucho (10 caracteres)
54	Suma de control del encabezamiento
	Espacio de trabajo de datos
67	Bit 1 encendido = EOF.Bit 2 encendido = no es arch. PRINT
68	Número registro (de 0 a 255)
69	Número de bytes en el archivo (máximo = 512)
71	Nombre archivo (10 caracteres)
81	Suma de control (bytes 67 a 80)
82	Inicio buffer de datos

0 – 82	82 – 594
ESPACIO DE TRABAJO	BUFFER DE DATOS (512 BYTES)

Kevin Jones

Estructura del canal del microdrive
El canal del microdrive ocupa 595 bytes de memoria. Su posición debe estar contenida en el registro IX después de abrir un fichero en el cartucho por medio del cód. de enganche 34

cho. Con un archivo de lectura, todo dato leído se destruirá si todavía se hallaba en el área de datos del canal. El canal es reclamado moviendo hacia arriba el área de memoria hasta STKEND que quedará inmediatamente debajo de ella. La llamada a la rutina se hace con la dirección de inicio de canal especificado y contenida en el registro IX. Se recomienda de nuevo salvar el otro par de registros HL antes de su empleo. Se dará un error si no hay espacio suficiente en el cartucho para cerrar el archivo.

• *Código de enganche 36:* Permite borrar un archivo del microdrive, tanto si es de datos como si es un programa. El nombre del archivo, su dirección y longitud, y el número del drive se establecen de la misma manera que con el código de enganche 34.

Ahora que conocemos cómo se abren y cierran los archivos será útil poder leer o escribir en ellos. Es posible acoplar a una corriente un canal del microdrive, pero la forma más fácil es hacer que un canal del microdrive sea el canal que lea o escriba (sustituyendo el teclado habitual y la corriente de la pantalla respectivamente).

El modo más sencillo de escribir datos en el canal del microdrive es usando el RST#10, después de haber establecido el canal del microdrive como canal de salida actual. Esto se hace poniendo 23633 y 23634 en la variable de sistema CURCHL, para que contenga la dirección de inicio del canal del microdrive. Esto se hace con una sola instrucción:

```
LD        (23633),IX
```

donde IX contiene la dirección de inicio del canal, según es retornada por el código de enganche 34. Hecho esto, el RST#10 escribirá bytes de datos en el área de canal del microdrive, y no en la pantalla. Esto es una nueva muestra de la absoluta versatilidad del sistema de canales y corrientes del Spectrum: un aspecto tan descuidado por el OS del ordenador. Como indicamos antes, tan pronto como el buffer de datos esté lleno, éstos se escribirán en el cartucho. Así, para escribir un archivo de nombre FRED en un cartucho, puede emplearse el siguiente fragmento de lenguaje máquina:

```
                ;escribe archivo FRED en el microdrive
3E01            ld    a,1            ;numero del drive
32D65C          ld    (23766),a      ;almacena el numero de drive
AF              xor   a              ;reg. a puesto a cero
32D75C          ld    (23767),a
212B29          ld    hl,name        ;toma el nombre en hl
22DC5C          ld    (23772),hl     ;almacena nombre archivo
210400          ld    hl,4           ;4 = longitud nombre
22DA5C          ld    (23778),hl     ;almacena long.nombre
D9              exx
E5              push hl;             ;salva hl'
D9              exx
CF              rst   #8
22              defb  34             ;cod enganche 34
DD22515C        ld    (23633),ix     ;CURCHL = canal drive
06FF            ld    b,255          ;archivo de 255
C5        loop: push  bc             ;salva contador
3E00            ld    a,CHAR         ;el usuario insertara
                                     ;la rutina para obtener
                                     ;de algun sitio un caracter
                                     ;dentro del reg a
D7              rst   #10            ;escribe en canal
C1              pop   bc             ;restaura contador
10F6            djnz  loop
CF              rst   #8
23              defb  35             ;cierra archivo
3E02            ld    a,2
CD0116          call  #1601          ;salida a pantalla
D9              exx
E1              pop   hl             ;restaura HL
D9              exx
FB              ei                   ;reactiva interrupciones
C9              ret
46524544 name:  defm  "FRED"         ;nombre archivo
```

Naturalmente, si usted abre un par de archivos, es necesario guardar las direcciones de inicio del canal de cada canal en algún área segura de la memoria (para poder ser utilizadas al cerrar los archivos, etc.). Nuestra rutina genera un archivo de 255 caracteres en el cartucho.

La lectura de datos desde este archivo es también inmediata. El archivo se abre como ya dijimos antes y se hace que CURCHL contenga la dirección de inicio del canal del microdrive. La rutina del Spectrum que está en #15E6 en la ROM principal se emplea ahora para leer un byte desde el canal. El byte leído se coloca en el registro A.

• *Código de enganche 37:* Si se llama, teniendo en IX la dirección de inicio del canal, esta rutina leerá el registro siguiente de un archivo, colocando los datos en el área de datos del canal. Si la lectura es correcta, el valor de CHREC se incrementa. Una vez leídos los datos de esta manera, repone, claro está, lo que existía antes en el área de datos del canal.

• *Código de enganche 38:* Es posible escribir datos en el cartucho mediante un código de enganche que lea el área de datos del canal en cuestión sobre la cinta. Esta rutina (el código de enganche 38) es entrada conteniendo el IX la dirección de inicio del canal. Observe que toda área de datos escrita en el cartucho con menos de 512 bytes puede ser considerada por el OS del Spectrum como un «sector libre» a menos que no esté marcado como registro *end of file*. Así, el código de enganche «Cerrar Archivo» se usará para escribir el último bloque de datos en la cinta si éste contiene menos de 512 bytes. Después de llamar al código de enganche 38, el contenido actual del área de datos es escrito en el archivo.

Una última rutina útil nos permitirá leer dentro de un archivo en un determinado registro. Es el código de enganche 39. En esta rutina el IX contiene la dirección de inicio del canal al que deseamos acceder, y CHREC contiene el número de registro seleccionado.

Existen otros códigos de enganche para el manejo de los archivos del microdrive, pero nos serán de menor utilidad en nuestra programación de cada día. El código de enganche 40 hace que puedan leerse los datos contenidos en un *sector* determinado del cartucho. CHREC contiene igualmente el número del sector al que deseamos acceder. El código de enganche 41 lee desde el cartucho el sector siguiente, y el código de enganche 42 escribe datos en el siguiente sector libre.

Dos últimos códigos de enganche debemos examinar, empleados para el manejo de los canales del microdrive. El código de enganche 43 producirá un mapa de canal y de cartucho. Pero, en la primera versión de la ROM de la Interface 1 existía un error en esta rutina que la convertía en un equivalente del código de enganche 34. En las versiones posteriores de esta ROM sí funciona. El código de enganche 44 borra un canal y el mapa asociado a él, cuya dirección de inicio está contenida en el registro IX.

Como punto final, diremos que se ha de tener mucho cuidado cuando se prueban los códigos de enganche, ya que el sistema de protección contra escritura que emplea el microdrive es sólo para *software*. Un fallo puede corromper el cartucho.

mi computer

Editorial Delta, S.A.



MI COMPUTER

SE PROLONGA

Con la aparición del fascículo 96 **"MI COMPUTER**, Curso Práctico del ordenador personal, el micro y el miniordenador" llega a su fin previsto, tras casi dos años de cita obligada y agradecida con nuestros lectores.

Pero el mundo de la informática está en contínua evolución y para responder al interés demostrado por nuestros lectores y a las numerosas cartas recibidas, hemos decidido enriquecer esta obra con **DOS NUEVOS VOLUMENES,** de 12 fascículos cada uno.

A lo largo de estos 24 nuevos fascículos, los lectores encontrarán la información más completa y actual sobre los últimos avances tecnológicos en el campo de la informática.

MI COMPUTER les ha introducido en el complejo mundo de la informática; la prolongación de la obra le convertirá en un experto.

